A reliable Turing machine

Ilir Çapuni
University of Montenegro
ilir@bu.edu

Peter Gács
Boston University
gacs@bu.edu

February 27, 2021

Abstract

We consider computations of a Turing machine subjected to noise. In every step, the action (the new state and the new content of the observed cell, the direction of the head movement) can differ from that prescribed by the transition function with a small probability (independently of previous such events). We construct a universal 1-tape Turing machine that for a low enough (constant) noise probability performs arbitrarily large computations. For this unavoidably, the input needs to be encoded—by a simple code depending on its size. The work uses a technique familiar from reliable cellular automata, complemented by some new ideas.

1 Introduction

This work addresses a question from the area of "reliable computation with unreliable components". A certain class of machines is chosen, (like a Boolean circuit, cellular automaton, Turing machine). It is specified what kind of faults (local in space and time) are allowed, and a machine of the given kind is built that—paying some price in performance—carries out essentially the same task as any given machine of the same kind without any faults would.

We confine attention to *transient*, *probabilistic* faults: the fault occurs at a given time but no component is damaged permanently, and faults occur independently of each other, with a bound on their probability. This in contrast to bounding the *number* of faults, allowing them to be set by an adversary.

Historically the first result of this kind is [11], which for each Boolean circuit C of size n constructs a new circuit C' of size $O(n \log n)$ that performs the same task as C with a (constant) high probability, even though each gate of C' is allowed to fail with some (constant) small probability.

Cellular automata as a model have several theoretical advantages over Boolean circuits, and results concerning reliable computation with them have interest also from a purely mathematical or physical point of view (non-ergodicity). The simple construction in [7] gives, for any 1-dimensional cellular automaton A a 3-dimensional cellular automaton A' that performs the same task as A with high probability, even though each cell of A' is allowed to fail in each time step with some constant small probability. (A drawback of this construction is the requirement of synchronization: all cells of A' must update simultaneously.) Reliable cellular automata in less than 3 dimensions can also be constructed (even without the synchrony requirement), but so far only at a steep increase in complexity (both of the construction and the proof). The first such result was [5], relying on some ideas proposed in [8].

Here, the reliability question will be considered for a *serial* computation model—Turing machine—as opposed to parallel ones like Boolean circuits or cellular automata. There is a single elementary processing unit (the *active* unit) interacting with a memory of unlimited size. The error model needs to be relaxed. Allowing each memory component to fail in each time step with constant probability makes, in the absence of parallelism, reliable computation seems impossible. Indeed, while in every step some constant fraction of the memory gets corrupted, the active unit can only correct a constant *number* of them per step.

In the relaxed model considered here, faults can affect only the operation of the active unit. More precisely at any given time the allowed operations of the machine are the usual ones: changing its state, writing to the observed tape cell, moving the head by a step left or right (or not at all). The transition table of the machine prescribes which action to take. So our fault model is the following.

Definition 1.1 Let *Noise* be a random subset of some set U. We will say that the distribution of *Noise* is ε*-bounded* if for every finite subset A we have

$$\mathsf{P}\{A \subseteq \mathit{Noise}\} \leq \varepsilon^{|A|}.$$

⌟

See [10] for an earlier use of this kind of restriction.

Definition 1.2 Let $\mathcal{C} = (C_1, C_2 \dots)$ be a random sequence of configurations of a Turing machine T with a given fixed transition table, with the property that for each time t, the C_{t+1} is obtained from C_t by one of the allowed operations. We say that a *fault* occurred at time t if the operation giving C_{t+1} from C_t is not obtained by the transition function. Let $Noise \subseteq \mathbb{Z}_+$ be the (random) set of faults in the sequence. We say that faults of the sequence $\mathcal{C}$ are ϵ*-bounded*, if the set *Noise* is. ⌟

The challenge for Turing machines is still significant, since even if only with small probability, occasionally a group of faults can put the head into the middle of a large segment of the tape rewritten in an arbitrarily "malicious" way. A method must be found to recover from all these situations.

Here we define a Turing machine that is reliable—under this fault model— in the same sense as the other models above. The construction and proof are similar in complexity to the ones for 1-dimensional cellular automata; however, we did not find a reduction to these earlier results. A natural idea is to let the Turing machine simulate a 1-dimensional cellular automaton, by having the head make large sweeps, and updating the tape as the simulated cellular automaton would. But apart from the issue of excessive delay, we did not find any simple way to guarantee the large sweeps in the presence of faults (where "simple" means not building some new hierarchy), even for some price paid in efficiency. So here we proceed "from scratch".

Many ideas used here are taken from [5] and [6], but hopefully in a somewhat simpler and more intuitive conceptual framework. Like [5] it confines all probability reasoning to a single lemma, deals on each level only with a numerical restriction on faults (of that level). On the other hand, like [6] it defines a series of generalized objects (generalized Turing machines here rather than generalized cellular automata), each one simulating the next in the series. In [6] a "trajectory" (central for defining the notion of simulation) was a random history whose distribution satisfies certain constraints (most of which are combinatorial). Here it is a single history satisfying only some combinatorial constraints.

The work [1] seems related by its title but is actually on another topic. The work [4] applies the self-simulation and hierarchical robustness technique developed for cellular automata in an interesting, but simpler setting. Several attempts at the chemical or biological implementation of universal computation have to deal with the issue of error-correction right away. In these cases generally the first issue is the faults occurring at the active site (the head). See [2, 9].

Our result can use any standard definition of 1-tape Turing machines whose tape alphabet contains some fixed "input-output alphabet" Σ; we will introduce one formally in Section 2.6. We will generally view a tape symbol as a tuple consisting of several *fields*. The notation

$$a.Output, a.Info$$

shows *Output* and *Info* as fields of tape cell state a. Combining the same field of all tape cells, we can talk about a *track* (say the *Output* track and *Info* track). For ease of spelling out a result, we consider only computations whose outcome is a single symbol, written into the *Output* field of tape position 1. It normally holds a special value—say $*$ —meaning *undefined*.

Block codes (as defined in Section 3.2 below) are specified by a pair (ψ_*, ψ^*) of encoding and decoding functions. In the theorem below, the input of the computation, of some length n, is broken up into blocks that are encoded by a block code that depends in some simple way on n. Its redundancy depends on the size of the input as a log power. The main result in the theorem below shows a Turing machine simulating a fault-free Turing machine computation in a fault-tolerant way. It is best to think of the simulated machine G as some universal Turing machine.

Theorem 1 *For any Turing machine G there are constants $\alpha_1, \alpha_2 > 0$, for each n a block code (φ_*, φ^*) of block size $O((\log n)^{\alpha_1})$, a fault bound $0 \leq \varepsilon < 1$ and a Turing machine M_1 with a function $a \mapsto a.Output$ defined on its alphabet, such that the following holds.*

Let M_1 start its work from the initial tape configuration $\varphi_*(x)$ with the head in position 0, running through a random sequence of configurations whose faults are ε-bounded in the sense of Definition 1.2. Suppose that at time t the machine G writes a value $y \neq *$ into the Output field of the cell at position 0. Then at any time greater than $t(\log t)^{\alpha_2}$, the tape symbol a of machine M_1 at position 0 will have $a.Output = y$ with probability at least $1 - O(\varepsilon)$.

2 Overview

The overview, but even the main text, is not separated completely into two parts: the definition of the Turing machine, followed by the proof of its reliability. The definition of the machine is certainly translatable (with a lot of tedious work) into just a Turing machine transition table (or *program*), but its complexity requires first to develop a *conceptual apparatus* behind it which is also used in the proof of reliability. We will try to indicate below at the beginning of each section whether it is devoted more to the program or more to the conceptual apparatus.

2.1 Isolated bursts of faults

(Introducing some basic elements of the *program*.) In [3] we defined a Turing machine M_1 that simulates "reliably" any other Turing machine even when it is subjected to isolated *bursts* of faults (that is a group of faults occurring in consecutive time steps) of constant size. We will use some of the ideas of [3], without relying directly on any of its details, and will add several new ideas. Let us give a brief overview of this machine M_1.

Each tape cell of the simulated machine M_2 will be represented by a block of some size Q called a *colony*, of the simulating machine M_1. Each step of M_2 will be simulated by a computation of M_1 called a *work period*. During this time, the head of M_1 makes a number of sweeps over the current colony-pair, decodes the represented cell symbols, then computes and encodes the new symbols, and finally moves the head to the new position of the head of M_2. The major processing steps will be carried out on a working track three times within one work period, recording the result onto separate tracks. The information track is changed only in a final majority vote.

The organization is controlled by a few key fields, for example a field called *Addr* showing the position of each cell in the colony, and a field *Sweep*, the number of the last sweep of the computation (along with its direction) that has been performed already. The most technical part is to protect this control information from faults. For example, a burst of faults of constant size can reverse the head in the middle of a sweep. To discover such structural disruptions locally before the head would go far in the wrong direction, it will make frequent short zigzags. A premature turn-back will be detected this way, triggering the healing procedure.

2.2 Hierarchy

(Starting to develop the *conceptual apparatus*.) In order to build a machine resisting faults occurring independently in each step with some small probability, we take the approach used for one-dimensional cellular automata. We aim at building a *hierarchy of simulations*: machine M_1 simulates machine M_2 which simulates machine M_3, and so on. Machine M_k has alphabet

$$\Sigma_k = \{0,1\}^{s_k}, \tag{2.1}$$

that is its tape cells have "capacity" s_k. All these machines should be implementable on a universal Turing machine with the same program (with an extra input, the number k denoting the level). For ease of analysis, we introduce the notion of *cell size*: level k has its own cell size B_k and block (colony) size Q_k with $B_1 = 1$, $B_{k+1} = B_k Q_k$. This allows locating a tape cell of M_k on the same interval where the cells of M_1 simulate it. One cell of machine M_{k+1} is simulated by a colony of machine M_k; so one cell of M_3 is simulated by $Q_1 Q_2$ cells of M_1. Further, one step of, say, machine M_3 is simulated by one work period of M_2 of, say, $O(Q_2^2)$ steps.

Per construction, machine M_1 can withstand bursts of faults with size $\leq \beta$ for some constant parameter β, separated by at least some constant γ work periods. It would be natural now to expect that machine M_1 can withstand also some *additional*, larger bursts of size $\leq \beta Q_1$ if those are separated by at least γ work periods of M_2. However, a *new obstacle* arises. Damage caused by a big burst of faults spans several colonies. The repair mechanism of machine M_1 outlined in Section 2.1 is too local to recover from such extensive damage, leaving the whole hierarchy endangered. So we add a new mechanism to M_1 that will just try to restore the colony structure of a large enough portion of the tape (of the extent of several colonies). The task of restoring the original information is left to higher levels (whose simulation now can continue).

All machines above M_1 in the hierarchy live only in simulation: the hardware is M_1. Moreover, the M_k with $k > 1$ will not be ordinary Turing machines, but *generalized* ones, with some new features seeming necessary in a simulated Turing machine: allowing for some "disordered" areas of the tape not obeying the transition function, and occasionally positive distance between neighboring tape cells.

A tricky issue is "forced self-simulation". Each machine M_k can be implemented on a universal machine using as inputs the pair (p,k) where p is the common program and k is the level. Eventually,

p will just be hard-wired into the definition of M_1, and therefore faults cannot corrupt it. While creating p for machine M_1, we want to make it simulate a machine M_2 that has the same program p. The method to achieve this is not really different from the one applied already in some of the cellular automata and tiling papers cited, and is related to the proof of Kleene's fixed-point theorem (also called the recursion theorem).

Forced self-simulation can give rise to an infinite sequence of simulations, achieving the needed robustness. But the simulation of M_{k+1} by M_k uses only certain tracks of the tape. Another track (which we will call *Rider*) will be set aside for simulating G. If the simulation of G on the *Rider* track of a colony-pair does not finish in a certain number of steps, a built-in mechanism will *lift* its tape content to the *Rider* field of the simulated cell-pair, allowing it to be continued in a colony-pair of the next level (with the corresponding higher reliability).

2.3 Structuring the noise

(Some definitions for analyzing the noise.) The set of faults in the noise model of the theorem is a set of points in time. It turns out more convenient to use an equivalent model: an ε-bounded *space-time* set of points. Let us make this statement more formal.

Lemma 2.1 *Let $\mathcal{C} = (C_1, C_2, \dots)$ be the random sequence of configurations of a Turing machine with an ε-bounded set of faults $Noise_1 \subseteq \mathbb{Z}_+$, as in Definition 1.2. Let $h(t)$ be the (random) position of the head at time t. Then the random set $Noise_2 = \{(h(t), t) : t \in Noise_1\}$. is an ε-bounded subset of $\mathbb{Z} \times \mathbb{Z}_+$.*

Proof. Let A be a finite subset of $\mathbb{Z} \times \mathbb{Z}_+$, and $A' = \{t : (p, t) \in A\}$. If $A \subseteq Noise_2$ then $A' \subseteq Noise_1$ and $|A'| = |A|$. Hence

$$\mathsf{P}\{A \subseteq Noise_2\} \leq \mathsf{P}\{A' \subseteq Noise_1\} \leq \varepsilon^{|A'|} = \varepsilon^{|A|}.$$

□

The construction outlined above counts with *bursts* (rectangles of space-time containing *Noise*) increasing in size and decreasing in frequency—which is a combinatorial set of constraints. To derive such constraints from the above probabilistic model the we stratify *Noise* as follows. We will have two series of parameters: $B_1 < B_2 < \cdots$ and $T_1 < T_2 < \cdots$. Here B_k is the size of cells of M_k as represented on the tape of M_1, and T_k is a bound on the time needed to simulate one step of M_k.

Here are some informal definitions (precise ones are given in Section 2.5). For some constants $\beta, \gamma > 1$, a *burst* of noise of type (a, b) is a space-time set that is essentially coverable by a $a \times b$. For an integer $k > 0$ it is of *level* k when it is of type $\beta(B_k \times T_k)$. It is *isolated* if it is essentially alone in a rectangle of type $\gamma(B_{k+1} \times T_{k+1})$ First we remove such isolated bursts of level 1, then of level 2 from the remaining set, and so on. It will be shown that with not too fast increasing sequences B_k, T_k, with probability 1, this sequence of operations eventually completely erases *Noise*: thus each fault belongs to a burst of "level" k for some k.

Machine M_k will concentrate only on correcting isolated bursts of level k and on restoring the framework allowing M_{k+1} to do its job. It can ignore the lower-level bursts and will need to work correctly only in the absence of higher-level bursts.

2.4 Difficulties

(Referring both to the program and to the concepts behind it.) We list here some of the main problems that the paper deals with, and some general ways in which they will be solved or avoided. Some more specific problems will be pointed out later, along with their solution.

Non-aligned colonies A large burst of faults in M_1 can modify the order of entire colonies or create new ones with gaps between them. To deal with this problem, machines M_k for $k > 1$ will be *generalized Turing machines*, allowing for non-adjacent cells.

Clean areas The tape of a generalized Turing machine will be divided, based on its content, into some areas called *clean*, the rest *disordered*. Clean areas will be essentially where the analysis can count on an existing underlying simulation, and where therefore the transition function is applicable. Noise can disorder the areas where it occurs.

Extending cleanness The predictability of the machine is decreased when the head enters into disorder. But the model still provides some "magical" properties helping to restore cleanness (in the absence of new noise):

A) escaping from any area in a bounded amount of time;

B) extension of clean intervals as the head passes in and out of them;

C) the cleaning of an interval when passed over a certain number of times.

While an area is cleaned, it will also be re-populated with cells. Their content is not important, what matters is the restoration of predictability.

Rebuilding The need to reproduce the cleaning properties in simulation is the main burden of the construction. The part of the program devoted to this is the rebuilding procedure, invoked when local repair fails. It reorganizes a part of the tape having the size of a few colonies.

2.5 Structuring the noise formally

(This purely mathematical section derives the combinatorial noise constraints from the probabilistic one.) If we modeled noise as a set of time points then a burst of faults would be a time interval of size βT_k and might affect a space interval as large as βT_k, covering many times more simulated cells of level k. Therefore we model noise as a set of space-time points; this does not change the independence assumption of the main theorem.

Definition 2.2 Let $\mathbf{r} = (r_1, r_2)$, $r_1, r_2 > 0$ be a two-dimensional nonnegative vector. A *rectangle of radius* $\mathbf{r}$ *centered* at point $\mathbf{x}$ is

$$B(\mathbf{x}, \mathbf{r}) = \{\mathbf{y} : |y_i - x_i| < r_i, i = 1, 2\}. \tag{2.2}$$

Let $E \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0}$ be a space-time set (to be considered our noise set). A point $\mathbf{x}$ of E is $(\mathbf{r}, \mathbf{r}^*)$-*isolated* if $E \cap B(\mathbf{x}, \mathbf{r}^*) \subseteq B(\mathbf{x}, \mathbf{r})$, that is all points of E that are $\mathbf{r}^*$-close to x are also $\mathbf{r}$-close. A set E is called $(\mathbf{r}, \mathbf{r}^*)$-*sparse* if each of its points is $(\mathbf{r}, \mathbf{r}^*)$-isolated. ⌟

The following lemma will justify talking about bursts of faults.

Lemma 2.3 *Suppose that the set E is $(\mathbf{r}, \mathbf{r}^*)$-sparse. Then in every rectangle U of size $r_1^* \times r_2^*$, $U \cap E$ is covered by some rectangle of size $r_1 \times r_2$.*

Proof. Let U be a rectangle of size $r_1^* \times r_2^*$, and suppose that $U \cap E$ is not covered by a rectangle of size $r_1 \times r_2$. Then either the horizontal projection of $E \cap U$ is larger than r_1 or the vertical one is larger than r_2, hence there is a pair of points $\mathbf{x}, \mathbf{y} \in E$ not covered by a single rectangle of size $r_1 \times r_2$. But then $U \subseteq B(\mathbf{x}, \mathbf{r}^*)$ and $\mathbf{y} \notin B(\mathbf{x}, \mathbf{r})$, hence $\mathbf{x}$ is not $(\mathbf{r}, \mathbf{r}^*)$-isolated contrary to the assumption of $(\mathbf{r}, \mathbf{r}^*)$-sparsity. □

Definition 2.4 Let $\gamma > 1$, $\beta > 4\gamma$ be parameters, and let

$$1 = B_1 < B_2 < \cdots, \quad 1 = T_1 < T_2 < \cdots,$$
$$T_{k+1}/T_k,\ B_{k+1}/B_k \geq 2\beta$$

be sequences of integers to be fixed later. For a space-time set $E \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0}$, let $E^{(1)} = E$. For $k > 1$ let $E^{(k+1)}$ be obtained by deleting from $E^{(k)}$ the $(\beta(B_k, T_k), \gamma(B_{k+1}, T_{k+1}))$-isolated points. Set E is *k-sparse* if $E^{(k+1)}$ is empty. It is simply *sparse* if $\bigcap_k E^{(k)} = \emptyset$. When $E = E^{(k)}$ and k is known then we will denote $E^{(k+1)}$ simply by E^*. ⌟

The following lemma connects the above defined sparsity notions to the requirement of small fault probability.

Lemma 2.5 (Sparsity) *Let $Q_k = B_{k+1}/B_k$, $U_k = T_{k+1}/T_k$, and*

$$\lim_{k \to \infty} \frac{\log Q_k U_k}{1.5^k} = 0. \tag{2.3}$$

For sufficiently small ε, for every $k \geq 1$ the following holds. Let $E \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0}$ be a random set that is ε-bounded as in Definition 1.2. Then for each point $\mathbf{x}$ and each k,

$$\mathsf{P}\{B(\mathbf{x}, (B_k, T_k)) \cap E^{(k)} \neq \emptyset\} < \varepsilon \cdot 2^{-1.5^{k-1}}.$$

As a consequence, the set E is sparse with probability 1.

This lemma allows a doubly exponentially increasing sequence U_k, resulting in relatively few simulation levels as a function of the computation time.

2.6 Generalized Turing machines

(This section, together with Section 2.8, introduces the key concepts used in the proof.) Let us recall that a one-tape Turing machine is defined by a finite set Γ of *internal states*, a finite alphabet Σ of *tape symbols*, a transition function δ, and possibly some distinguished states and tape symbols. At any time, the head is at some integer position h, and is observing the tape symbol $A(h)$. The meaning of $\delta(a,q) = (a',q',d)$ is that if $A(h) = a$ and the state is q then the $A(h)$ will be rewritten as a' and h will change to $h+d$.

We will use a model that is slightly different, but has clearly the same expressing power. (Its advantage is that it is a little more convenient to describe its simulations.) There are no internal states, but the head observes and modifies a *pair* of neighboring tape cells at a time; in fact, we imagine it to be positioned between these two cells called the *current cell-pair*. The *current cell* is the left element of this pair. Thus, a Turing machine is defined as (Σ,τ) where the tape alphabet Σ contains at least the distinguished symbols $\llcorner, 0, 1$ where $\llcorner$ is called the *blank symbol*. The *transition function* is $\tau\colon \Sigma^2 \to \Sigma^2 \times \{-1,1\}$. A *configuration* is a pair $\xi = (A,h) = (\xi.\text{tape}, \xi.\text{pos})$ where $h \in \mathbb{Z}$ is the *current* (or *observed)* head position, (between cells h and $h+1$), and $A \in \Sigma^{\mathbb{Z}}$ is the *tape content*, or *tape configuration*: in cell p, the tape contains the symbol $A(p)$. Though the tape alphabet may contain non-binary symbols, we will restrict input and output to binary. The tape is blank at all but finitely many positions.

As the head observes the pair of tape cells with content $\mathbf{a} = (a_0, a_1)$ at positions h, $h+1$ denote $(\mathbf{a}', d) = \tau(\mathbf{a})$. The transition τ will change tape content at positions h, $h+1$ to a_0', a_1', and move the head to tape position to $h+d$. A *fault* occurs at time t if the output $(\mathbf{a}', d)$ of the transition function at this time is replaced with some other value (which then defines the next configuration).

The machines that occur in simulation will be a generalized version of the above model, allowing non-adjacent cells and areas called "disordered" in which the transition function is non-applicable. A mere convenience feature is two integer parameters: the cell body size $B \geq 1$ and and an upper bound $T \geq 1$ on the transition time. These allow placing all the different Turing machines in a hierarchy of simulations onto the same space line and the same time line.

Definition 2.6 (Generalized Turing machine) A *generalized Turing machine M* is defined by a tuple

$$(\Sigma, \tau, B, T, \pi), \tag{2.4}$$

where Σ is the *alphabet*, and

$$\tau : \Sigma^2 \times \{\text{True}, \mathit{False}\} \to \Sigma^2 \times \{-1,1\}.$$

is the *transition function*. In $\tau(a,b,\alpha)$ the argument α is True if the pair of observed cells is adjacent (no gap between them), and False otherwise. Among the elements of the tape alphabet we distinguish the input-output element 0, 1, a special symbol Vac and a subset New. Vac plays the role of a blank symbol (the absence of a cell), and the state of newly created cells is in New.

In the definition of tape configurations we will add another symbol: Bad will mark *disordered* areas of the tape. The transition function τ has no inputs or outputs that are Bad or Vac. Let $\tau(a, b, \alpha) = (a', b', d)$. We can have $a' \in$ New or $b' \in$ New only if $a, b \notin$ New and $\alpha =$ False, that is the observed cells are not adjacent. Even then we can have $a' \in$ New only if $d = -1$ and $a' \in$ New only if $d = 1$. The effect of the transition function on configurations will be explained in Definition 2.9.

The integer π will play the role of the number of passes needed to clean an area (see below). ⌟

A formal definition of a configuration of a generalized Turing machine is given in Section 3.4, though it is essentially defined by the tape content A and the head position. A point p is *clean* if $A(p) \neq$ Bad. A set of points is *clean* if it consists of clean points. We say that there is a *cell* at a position $p \in \mathbb{Z}$ if the interval $p + [0, B)$ is clean and $A(p) \neq$ Vac. In this case, we call the interval $p + [0, B)$ the *body* of this cell. Cells must be at distance $\geq B$ from each other, that is their bodies must not intersect. If their bodies are at a distance $< B$ from each other (with a clean interval containing both) then they are called *neighbors*. They are called *adjacent* if this distance is 0.

A sequence of configurations conceivable as a computation will be called a "history". For standard Turing machines, the histories that obey the transition function could be called "trajectories". For generalized Turing machines the definition of trajectories is more complex; it allows some limited violations of the transition function, while providing the mechanisms for eliminating disorder. Let $Noise \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0}$ denote the set of space-time points at which faults occur. Section 2.8 below will define a certain subset of possible histories called *trajectories*. In order to motivate their choice, we first introduce the notion of simulation.

2.7 Simulation

(Here we introduce the notion of simulation used in the proof. It relies on the concept of trajectories defined in Section 2.8, but also motivates it.) Until this moment, we used the term "simulation" informally, to denote a correspondence between configurations of two machines which remains preserved during the computation. In the formal definition, this correspondence will essentially be a code $\varphi = (\varphi_*, \varphi^*)$. The *decoding* part of the code is the more important. We want to say that machine M_1 simulates machine M_2 via simulation φ if whenever $(\eta, Noise)$ is a trajectory of M_1 then $(\eta^*, Noise^*)$, defined by $\eta^*(\cdot, t) = \varphi^*(\eta(\cdot, t))$, is a trajectory of M_1. Here, $Noise^*$ is computed by the residue operation (deleting isolated bursts) as in Definition 2.4. We will make, however, two refinements. First, we require the above condition only for those η for which the initial configuration $\eta(\cdot, 0)$ has been obtained by encoding, that is it has the form $\eta(\cdot, 0) = \varphi_*(\xi)$. Second, to avoid the transitional ambiguities in a history, we define the simulation decoding as a mapping Φ^* between *histories*, not just configurations: $\Phi^*(\eta, Noise) = (\eta^*, Noise^*)$.

Definition 2.7 (Simulation) Let M_1, M_2 be two generalized Turing machines, and let $\varphi_* : \text{Configs}_{M_2} \to \text{Configs}_{M_1}$ be a mapping from configurations of M_2 to those of M_1, such that it maps starting configurations into starting configurations. Let $\Phi^* : \text{Histories}_{M_1} \to \text{Histories}_{M_2}$ be a map-

ping. The pair (φ_*, Φ^*) is called a *simulation* (of M_2 by M_1) if for every trajectory (η, Noise) of M_1 with initial configuration $\eta(\cdot, 0) = \varphi_*(\xi)$, the history $(\eta^*, \mathit{Noise}^*) = \Phi^*(\eta, \mathit{Noise})$ is a trajectory of machine M_2. ⌟

In the noise-free case it is easy to find examples of simulations. However, in the cases with noise, finding any nontrivial example is a challenge, and depends on a careful definition of trajectories for generalized Turing machines.

2.8 Trajectories

(This completes the definition of the central concept of the proof—modulo the natural definitions spelled out in Section 3.4.) The set of trajectories of a generalized Turing machine M is defined in terms of constraints imposed on the fault-free parts of a history. We discuss these properties first informally.

Transition Function This property says (in more precise terms) that in a clean area, the transition function is obeyed.

The Spill Bound limits the extent to which a disordered interval can spread while the head is in it.

Escape limits the time for which the head can be trapped in a small area.

Attack Cleaning erodes disorder as the head repeatedly enters and leaves it.

Pass Cleaning cleans the interior of an interval if the head passes over it enough times.

Remark 2.8 The Pass Cleaning property is only used in the proof of the other trajectory properties (and itself) for the simulated trajectory η^*. Without it, it could happen that the head slides over a large disordered area many times (staying inside) and, occasionally, extends it by new faults. ⌟

The definition below depends on the notions of current cell-pair, switch and dwell period given in in Section 3.4, but should be understandable as it is.

Definition 2.9 Suppose that at times t' before a switching time t but after any previous switch, the current cell-pair (x, y) has state $\mathbf{a} = (a, b)$. Let $(a', b', d) = \tau(a, b, \alpha)$, where $\alpha =$ True if the cell-pair is adjacent and False otherwise. Let u, v be the states of the cells x, y after the transition, and let x', y' be the new current cell pair. We say that the switch is *dictated by the transition function* if the following holds. We state the conditions for $d = 1$, the case $d = -1$ is analogous.

- $u = a'$.
- Suppose $b' \notin$ New; then $v = b'$, $x' = y$. If cell y has a neighbor z on the right then $y' = z$. Else a new adjacent neighbor z is created on the right of y with a state in New, and again $y' = z$.
- If $b' \in$ New (in which case $\alpha =$ False and x, y are not adjacent) then $v =$ Vac (cell y is erased), $x' = x$, and a cell y' adjacent to x on the right is created with state b'. We will say that cell y is *replaced* with the new cell y'.

⌟

As a consequence of this definition, new cells are created automatically when the head would step onto a vacant area, and whenever a cell is killed another one is created automatically in a place overlapping with its body.

We will use the following constants:

$$c_{\text{spill}} = 2,\ c_{\text{marg}} = 13,\ c_{\text{pass}} = 3c_{\text{marg}},\ c_{\text{esc}} = 7. \tag{2.5}$$

, and we will assume

$$\beta \geq c_{\text{pass}}. \tag{2.6}$$

Definition 2.10 For a set K on the line, and some real c let us define its *c-interior* $\text{Int}(K, c)$ as the set of those points of K that are at a distance $\geq c$ from its complement. For an interval $K = [a, b)$, we this is $[a + c, b - c)$. In this case, will use it also with negative c; then "interior" is really an extended neighborhood of I. ⌟

In the following definition, it is important to keep in mind the difference between noise and disorder. A noise-free space-time rectangle can very well contain disordered areas on the tape.

Definition 2.11 (Trajectory) A history (η, Noise) of a generalized Turing machine (2.4) with $\eta(t) = (A(t), h(t), \hat{\mathbf{h}}(t))$ is called a *trajectory* of M if the following conditions hold, in any noise-free space-time interval $I \times J$.

Transition Function Consider a switch, where the current cell-pair $\hat{\mathbf{h}}$ is inside a clean area, by a distance of at least $2.5B$. Then the new state of the current cell-pair and the direction towards the new current head position are dictated by the transition function. We further require that if the head moved right and $\hat{\mathbf{h}}$ has no right neighbor cell then a new adjacent right neighbor cell is created. Similarly in the left direction.

The only change on the tape occurs on the interval enclosing the new and old current cells. Further, the length of the dwell period before the switch is bounded by T.

Spill Bound A clean interval can shrink by at most $c_{\text{spill}}B$ while it does not contain the head.

Escape The head will leave any interval of size $\leq \lambda B$ with $1 \leq \lambda \leq 3\beta$ within time $c_{\text{esc}}\lambda^2 T$.

Attack Cleaning Suppose that the current cell-pair is at the end of a clean interval $[a, b)$ of size $\geq (c_{\text{marg}} + 1)B$, with the head at position x. Suppose further that the transition function directs the head right. Then by the time the head comes back to $x - c_{\text{marg}}B$, the clean interval extends at least to $[a, b + B/2)$. Similarly when "left" and "right" are interchanged.

Pass Cleaning Suppose that a path P makes at least π passes over an interval I of size $\leq c_{\text{pass}}B$. Then at some time during P the interior $\text{Int}(I, c_{\text{marg}}B)$ of I becomes clean.

⌟

Recall that we will have a hierarchy of simulations $M_1 \longrightarrow M_2 \longrightarrow \cdots$ where machine M_k simulates machine M_{k+1}. Our construction will set $\pi = 8k + O(1)$ for M_k. This can be interpreted as saying that each 8 passes will raise the "organization level".

2.9 Scale-up

Above, we have set up the conceptual structure of the construction and the proof. Here are some of the parameters:

Definition 2.12 Let $Q_k = c_1 \cdot 2^{1.2^k}$, $U_k = Q_k^3 = c_1^3 \cdot 2^{3 \cdot 1.2^k}$, $\pi_k = 8k + c_2$ for appropriate constants $c_1, c_2 > 0$. These sequences clearly satisfy (2.3), and define $B_k = B_1 \prod_{i<k} Q_i$, $T_k = T_1 \prod_{i<k} U_i$. ⌟

What remains is the definition of the simulation program and the decoding Φ^*, and the proof that with this program, the properties of a trajectory (η, Noise) of machine $M = M_k$ imply that the history $\Phi^*(\eta, \mathit{Noise}) = (\eta^*, \mathit{Noise}^*)$ obeys the same trajectory requirements on the next level. The program is described in Sections 4-5. The most combinatorially complex part of the proof of trajectory properties is in the key Section 7, proving the trajectory properties related to bounding and eliminating disorder. Section 8 wraps up the proof of the main theorem.

3 Some formal details

We give here some details that were postponed from the overview section.

3.1 Examples

The following examples show the difficulties responsible for various complexities of the construction and proof. Some of them may not be completely understandable without the details of the following program. You can skip these examples safely, and return to them later when wondering about the motivation for some feature.

Example 3.1 (Need for zigging) When the head works in a colony of machine M_1 performing a simulation of a cell of machine M_2, it works in sweeps across the colony. But a small burst of faults could reverse the head in the middle of a sweep, leaving it uncompleted. This way local faults could create non-local inconsistency.

⌟

To handle the problem of Example 3.1, the head will proceed in zigzags: every step advancing the head in the simulation is followed by Z steps of going backward and forward again (with parameter Z chosen appropriately), checking consistency (and starting a healing process if necessary). This will also enable the head to progress into a large disordered area, preventing it from being fooled repeatedly into going away, this way solving an even more serious problem.

Example 3.2 (Need for feathering) Some big noise can create a number of intervals $I_1, I_2, \ldots, I_n$ consisting of colonies of machine M_1, each interval with its own simulated head, where the neighboring intervals are in no relation to each other. When the head is about to return from the end of I_k (never even to zig beyond it, see the discussion after Example 3.1), a small burst of faults can carry it over to I_{k+1} where the situation may be symmetric: it will continue the simulation that I_{k+1} is performing. (The rightmost colony of I_k and the leftmost colony of I_{k+1} need not be complete: what matters is only that the simulation in I_k would not bring the head beyond its right end, and the simulation in I_{k+1} would not bring the head beyond its left end.)

The head can be similarly captured to I_{k+2}, then much later back from I_{k+1} to I_k, and so on. This way the restoration of structure in M_2 may be delayed too long. ⌟

The device by which we will mitigate the effect of this kind of capturing is another property of the the movement of the head which will call *feathering*: if the head turns back from a tape cell then next time it must go beyond. This requires a number of adjustments to the program (see later).

Example 3.3 (Two slides over disorder) This example shows the possibility for the head to slide twice over disorder without cleaning it.

Consider two levels of simulation as outlined in Section 2.2: machine M_1 simulates M_2 which simulates M_3. The tape of M_1 is subdivided into colonies of size Q_1. A burst of faults on level 1 has size $O(1)$, while a burst on level 2 has size $O(Q_1)$.

Suppose that M_1 is performing a simulation in colony C_0. An earlier big burst may have created a large interval D of disorder on the right of C_0, even reaching into C_0. For the moment, let C_0 be called a *victim* colony. Assume that the left edge of D represents the last stage of a transfer operation to the right neighbor colony C_0+Q_1. When the head, while performing its work in C_0, moves close to its right end, a small burst may carry it over into D. There it will be "captured", and continue the (unintended) right transfer operation. This can carry the head, over several successful colony simulations in D, to some victim colony C_1 on the right from which it will be captured to the right similarly. This can continue over new and new victim colonies C_i (with enough space between them to allow for new faults to occur), all the way inside the disorder D. So the M_2 cells in D will fail to simulate M_3.

After a while the head may return to the left in D (performing the simulations in its colonies). When it gets at the right end of a victim colony C_i, a burst of faults might move it back there. There is a case when C_i now can just continue its simulation and then send the head further left: when before the head was captured on its right, it was in the last stage of simulating a left turn of the head of machine M_2.

In summary, a big burst of faults can create a disordered area D which can capture the head and on which the head can slide forward and back without recreating any level of organization beyond the second one. ⌟

Example 3.4 (Many slides over disorder) Let us describe a certain "organization" of a disordered area in which an unbounded number of passes may be required to restore order. For some $n < 0$,

let the cells of M_1 at positions $x_{-Q_1}, \ldots, x_n$, where $x_{i+1} = x_i + B_1$, represent part of a healthy colony $C(x_{-Q_1})$ starting at x_{-Q_1}, where x_n is the rightmost cell of $C(x_{-Q_1})$ to which the head would come in the last sweep before the simulation will move to the *left* neighbor colony $C(x_{-2Q_1})$. Let them be followed by cells $x_{n+1}, \ldots, x_{Q_1-1}, \ldots$ which represent the last sweep of a transfer operation to the *right* neighbor colony $C(x_0)$. If the head is in cell x_n, a burst of faults can transfer it to x_{n+1}. The cell state of M_2 simulated by $C(x_{-Q_1})$ need to be in *no relation* to the cell state of M_2 simulated by $C(x_0)$. This was a capture of the head by a burst of M_1 across the point 0, to the right.

We can repeat the capture scenario, say around points iQ_1Q_2 for $i = 1, 2, \ldots$, and this way cells of M_3 simulated by M_2 (simulated by M_1) can be defined arbitrarily, with no consistency needed between any two neighbors. (We did not write iQ_1 just in case bursts are not allowed in neighboring colonies.) In particular, we can define them to implement a *leftward* capture scenario via level 3 bursts at points $iQ_1Q_2Q_3Q_4$, allowing to simulate arbitrary cells of M_5 with no consistency requirement between neighbors. So M_5 could again implement a rightward capture scenario, and so on. In summary, a malicious arrangement of disorder and noise allows k passes after which the level of organization is still limited to level $2k + 1$. ⌟

Our construction will ensure that, on the other hand, $O(k)$ passes (free of k-level noise) will restore organization to level k. This property of the construction will be incorporated into our definition of a generalized Turing machines as the "magical" property (C) above.

3.2 Codes

The input of our computation will be encoded by some error-correcting code, to defend against the possibility of losing information even at the first reading.

Definition 3.5 (Codes) Let Σ_1, Σ_2 be two finite alphabets. A *block code* is given by a positive integer Q—called the *block size*—and a pair of functions

$$\psi_* : \Sigma_2 \to \Sigma_1^Q, \quad \psi^* : \Sigma_1^Q \to \Sigma_2$$

with the property $\psi^*(\psi_*(x)) = x$. Here ψ_* is the encoding function (possibly introducing redundancy) and ψ^* is the decoding function (possibly correcting errors). The code is extended to (finite or infinite) strings by encoding each letter individually:

$$\psi_*(x_1, \ldots, x_n) = \psi_*(x_1) \cdots \psi_*(x_n).$$

⌟

3.3 Proof of the sparsity lemma

Proof of Lemma 2.5. The proof uses slightly more notation than it would if we simply assumed independence of faults at different space-time sites, but it is essentially the same.

Let $\mathcal{E}_k(\mathbf{x})$ be the event $B(\mathbf{x},(B_k,T_k))\cap E^{(k)}\neq\emptyset$. Let $\mathcal{M}=\mathcal{M}_k(\mathbf{x})$ be the set of minimal sets $A\subseteq\mathbb{Z}\times\mathbb{Z}_+$ with $A\subseteq E\Rightarrow\mathcal{E}_k(\mathbf{x})$.

Claim 3.6 *Each set in $\mathcal{M}_k(\mathbf{x})$ is contained in $B(\mathbf{x},2\gamma(B_k,T_k))$.*

Proof. The statement is clearly true for $k=1$. Suppose it is true for k, let us prove it for $k+1$. The event $\mathcal{E}_{k+1}(\mathbf{x})$ holds if and only if $\mathbf{x}\in E$ and there is some point $\mathbf{y}$ in $E^{(k)}\cap B(\mathbf{x},\gamma(B_{k+},T_{k+1}))\setminus B(\mathbf{x},\beta(B_k,T_k))$. Then by the inductive assumption, $A\subseteq E$ for some set $A\subseteq B(\mathbf{y},2\gamma(B_k,T_k))$, with the property $A\subseteq E\Rightarrow\mathcal{E}_k(\mathbf{y})$. Then $A\cup\{\mathbf{x}\}\subseteq E\Rightarrow\mathcal{E}_{k+1}(\mathbf{x})$. Also

$$A\subseteq B(\mathbf{x},(\gamma B_{k+1}+2\gamma B_k,\gamma T_{k+1}+2\gamma T_k))\subseteq B(\mathbf{x},2\gamma(B_{k+1},T_{k+1})),$$

since $T_{k+1}/T_k>2$, $B_{k+1}/B_k>2$. □

Let $f_k(\mathbf{x})=\sum_{A\in\mathcal{M}}\varepsilon^{|A|}$. By the union bound we have $\mathsf{P}(\mathcal{E}_k(x))\le f_k(\mathbf{x})$.

Let $p_k=\varepsilon\cdot 2^{-1.5^{k-1}}$. We will prove $f_k(\mathbf{x})<p_k$ by induction. For $k=1$, rectangles $B(\mathbf{x}_i,(B_1,T_1))$ have size 1, so by the ε-boundedness, $f_1(\mathbf{x})<\varepsilon$. Assume that the statement holds for k, we will prove it for $k+1$.

Suppose $\mathbf{y}\in E^{(k)}\cap B(\mathbf{x},(B_{k+1},T_{k+1}))$. According to the definition of $E^{(k)}$, there is a point

$$\mathbf{z}\in B(\mathbf{y},\gamma(B_{k+1},T_{k+1}))\cap E^{(k)}\setminus B(\mathbf{y},\beta(B_k,T_k)). \tag{3.1}$$

Consider a standard partition of space-time into rectangles $K_p=B(\mathbf{c}_p,(B_k,T_k))$. Let

$$I=\{p:K_p\cap B(\mathbf{x},\gamma(B_{k+1},T_{k+1}))\neq\emptyset\}.$$

We are only interested in rectangles K_p with $p\in I$. Let

$$K'_p=B(\mathbf{c}_p,((2\gamma+1)B_k,(2\gamma+1)T_k)).$$

If K_i,K_j are the rectangles in this partition containing $\mathbf{y}$ and $\mathbf{z}$, then $K'_i\cap K'_j=\emptyset$. This follows from the fact that $|y_1-z_1|>\beta B_k$, $|y_2-z_2|>\beta T_k$, and $\beta>4\gamma$ in Definition 2.4. By the above Claim, the event $\mathbf{y}\in E^k$ can be written as $\bigcup_{A\in\mathcal{M}_k(\mathbf{y})}\{A\subseteq E\}$ where $A\subseteq B(\mathbf{y},2\gamma(B_k,T_k))\subseteq K'_i$ for each $A\in\mathcal{M}_k(\mathbf{y})$. Similarly for $\mathbf{z}$. Let $\mathcal{M}(i)=\bigcup_{\mathbf{y}\in K_i}\mathcal{M}_k(\mathbf{y})$, then each set $A\in\mathcal{M}(i)$ is in K'_i. The disjointness of K'_i and K'_j and the inductive assumption implies

$$\begin{aligned}f_{k+1}(\mathbf{x})&\le\sum_{i,j\in I,K'_i\cap K'_j=\emptyset}\ \sum_{A\in\mathcal{M}(i),A'\in\mathcal{M}(j)}\varepsilon^{|A|+|A'|}=\sum_{i,j\in I,K'_i\cap K'_j=\emptyset}f_k(\mathbf{c}_i)f_k(\mathbf{c}_j)\\&\le|I|^2p_k^2=|I|^2\varepsilon^2 2^{-1.5^k}\cdot 2^{-0.5\cdot 1.5^{k-1}}=p_{k+1}\varepsilon|I|^2 2^{-0.5\cdot 1.5^{k-1}}.\end{aligned}\tag{3.2}$$

We have $|I|\le(2\gamma Q_k+1)(2\gamma U_k+1)$. Since $\lim_k\frac{\log U_kQ_k}{1.5^k}=0$, the multiplier of p_{k+1} in (3.2) is ≤ 1 for sufficiently small ε. □

3.4 Configuration, history

A configuration, as defined below, contains a pair of positions $\hat{\mathbf{h}} = (\hat{h}_0, \hat{h}_1)$ called the *current cell-pair*: In difference to the Turing machines of Section 2.6, the position of the head may not be exactly between the current cells: this allows the model to fit into the framework where a generalized Turing machine M^* is simulated by some (possibly generalized) Turing machine M. The head h of M—made equal to that of M^*—may oscillate inside and around the current cell-pair of M^*.

Definition 3.7 (Configuration) A *configuration* ξ of a generalized Turing machine (2.4) is a tuple

$$(A, h, \hat{\mathbf{h}}) = (\xi.\text{tape}, \xi.\text{pos}, (\xi.\text{cur-cell}_0, \xi.\text{cur-cell}_1))$$

where $A : \mathbb{Z} \to \Sigma$ is the tape, $h \in \mathbb{Z}$ is the head position, $\hat{\mathbf{h}} \in \mathbb{Z}^2$ is the current cell-pair. We have $A(p) = \text{Vac}$ in all but finitely many positions p. Whenever the interval $h + [-4B, 4B)$ is clean the current cell-pair must be within it. Let

$$\text{Configs}_M$$

denote the set of all possible configurations of a Turing machine M. ⌟

The above definitions can be localized to define a configuration over a space interval I containing the head.

Definition 3.8 (History) For a generalized Turing machine (2.4), consider a sequence $\eta = (\eta(0, \cdot), \eta(1, \cdot), \dots)$, along with a noise set *Noise*. Let $h(t) = \eta(t, \cdot).\text{pos}$ be the position of head.

A *switching time* is a noise-free time when any part of η other than $h(t)$ changes. A *dwell period* is the interval between any consecutive pair of switching times with the property that the space-time rectangle between them and containing the head is clean and noiseless.

The pair $(\eta, \textit{Noise})$ will be called a *history* of machine M if the following conditions hold.

- $|h(t) - h(t')| \le |t' - t|$.
- In two consecutive configurations, the content $A(p, t)$ of the positions p not in $h(t) + [-2B, 2B)$ remains the same.
- At each noise-free switching time the head is on the new current cell-pair: $\hat{h}_0(t) = h(t)$. (In particular, when at a switching time a current cell becomes Vac, the head must already be elsewhere.)
- The length of dwell periods is at most T.

The above definition can be localized to define a history $I \times J$ containing the head. Let

$$\text{Histories}_M$$

denote the set of all possible histories of M. ⌟

3.5 Hierarchical codes

Recall the notion of a code in Definition 3.5.

Definition 3.9 (Code on configurations) Consider two generalized Turing machines M_1, M_2 with the corresponding alphabets and transition functions, where B_2/B_1 is an integer denoted $Q = Q_1$. Assume that a block code $\psi_* : \Sigma_2 \to \Sigma_1^Q$ is given, with an appropriate decoding function, ψ^*. Symbol $a \in \Sigma_2$, is interpreted as the content of some tape square.

This block code gives rise to a *code on configurations*, that is a pair of functions

$$\varphi_* : \text{Configs}_{M_2} \to \text{Configs}_{M_1}, \quad \varphi^* : \text{Configs}_{M_1} \to \text{Configs}_{M_2}$$

that encodes some (initial) configurations ξ of M_2 into configurations of M_1: each cell of M_2 is encoded into a colony of M_1 occupying the same interval. Formally, assuming $\xi.\text{cur-cell}_j = \xi.\text{pos} + (j-1)B_2$, $j = 0, 1$ we set $\varphi_*(\xi).\text{pos} = \xi.\text{pos}$, $\varphi_*(\xi).\text{cur-cell}_j = \varphi_*(\xi).\text{pos} + (j-1)B_2$, and

$$\varphi_*(\xi).\text{tape}(iB_2, iB_2 + B_1, \ldots, (i+1)B_2 - B_1) = \psi_*(\xi.\text{tape}(i)).$$

⌟

Definition 3.10 (Hierarchical code) For $k \geq 1$, let Σ_k be an alphabet, of a generalized Turing machine M_k. Let $Q_k = B_{k+1}/B_k$ be an integer (viewed as colony size), let φ_k be a code on configurations defined by a block code

$$\psi_k : \Sigma_{k+1} \to \Sigma_k^{Q_k}$$

as in Definition 3.9. The sequence (Σ_k, φ_k), $(k \geq 1)$, is called a *hierarchical code*. For this hierarchical code, configuration ξ^1 of M_1 is called a *hierarchical code configuration* of height k if a sequence of configurations $\xi^2, \xi^3, \ldots, \xi^k$ of $M_2, M_3, \ldots, M^k$ exists with

$$\xi^i = \varphi_{*i}(\xi^{i+1})$$

for all i. If we are also given a sequence of mappings $\Phi_1^*, \Phi_2^*, \ldots$ such that for each i, the pair (φ_{i*}, Φ_i^*), is a simulation of M_{i+1} by M_i then we have a *hierarchy of simulations* of height k. ⌟

We will construct a hierarchy of simulations whose height grows during the computation—by a mechanism to be described later.

4 Simulation structure

In what follows we will describe the program of the reliable Turing machine: a hierarchical simulation in which simultaneously each M_{k+1} is simulated by M_k, with an added mechanism to raise the height of the hierarchy when needed. Most of the time, we will write $M = M_k$, $M^* = M_{k+1}$. Ideally, cells will

be grouped into colonies of size $Q = B^*/B$. Simulating one step of M^* takes a sequence of steps of M constituting a *work period*. Machine M will perform the simulation as long as the noise in which it operates is $(\beta(B,T), \gamma(B^*,T^*))$-sparse (as in Definition 2.2). This means, informally, that a burst of noise affects at most β consecutive tape cells, and there is at most one burst in any γ neighboring work periods. A design goal for the program is to correct a burst within space much smaller than a colony.

Modes were introduced in Section 4.2. Ordinary simulation proceeds in the normal mode. To see whether consistency, that is the basic tape pattern supporting simulation, is broken somewhere, a very local precaution will be taken in each step: each step will check whether the current cell-pair is allowed in a healthy configuration. If not then the mode of the current pair will be changed into *healing*. We will also say that *alarm* will be called. On the other hand, the state may enter into *rebuilding* mode on some indications that healing fails.

4.1 Error-correcting code

Let us add error-correcting features to the block codes introduced in Definition 3.5.

Definition 4.1 (Error-correcting code) A block code is (β, t)-*burst-error-correcting*, if for all $x \in \Sigma_2$, $y \in \Sigma_1^Q$ we have $\psi^*(y) = x$ whenever y differs from $\psi_*(x)$ in at most t intervals of size $\leq \beta$. For such a code, we will say that a word $y \in \Sigma_1^Q$ is *r-compliant* if it differs from a codeword of the code by at most r intervals of size $\leq \beta$. ⌟

Example 4.2 (Repetition code) Suppose that $Q \geq 3\beta$ is divisible by 3, $\Sigma_2 = \Sigma_1^{Q/3}$, $\psi_*(x) = xxx$. If $y = y(1)\dots y(Q)$, then $x = \psi^*(y)$ is defined by $x(i) = \mathrm{maj}(y(i), y(i+Q/3), y+2Q/3)$. For all $\beta \leq Q/3$, this is a $(\beta, 1)$-burst-error-correcting code. If we repeat 5 times instead of 3, we get a $(\beta, 2)$-burst-error-correcting code. ⌟

Example 4.3 (Reed-Solomon code) There are much more efficient such codes than just repetition. One, based on the Reed-Solomon code, is outlined in Example 4.6 of [6]. If each symbol of the code has l bits then the code can be up to 2^l symbols long. Only $2t\beta$ of its symbols need to be redundant in order to correct t faults of length β. ⌟

Consider a (generalized) Turing machine (Σ, τ) simulating some Turing machine (Σ^*, τ^*). We will assume that the alphabet Σ^* is a subset of the set of binary strings $\{0,1\}^l$ for some $l < Q$.

We will store the coded information in the interior of the colony, since it is more exposed to errors near the boundaries.

Definition 4.4 Let

$$PadLen$$

be a parameter to be defined later (in Definition 4.13). A cell belongs to the *interior* of a colony spanning an interval I if it is in $\mathrm{Int}(I, PadLen)$ (with the interior as in Definition 2.10). ⌟

Let (υ_*, υ^*) be a $(\beta, 2)$-burst-error-correcting block code with

$$\upsilon_* : \{0,1\}^l \cup \{\emptyset\} \to \{0,1\}^{(Q-2\cdot PadLen)B}.$$

We could use, for example, the repetition code of Example 4.2. Other codes are also appropriate, but we require that there are some fixed Turing machines Encode and Decode computing them:

$$\upsilon_*(x) = \text{Encode}(x), \quad \upsilon^*(y) = \text{Decode}(y).$$

Also, these programs must work in quadratic time and linear space on a one-tape Turing machine (as the repetition code certainly does).

Recall that our Turing machine has some special states, among others: $0, 1, \text{new}_0, \text{new}_1$. We require that at least some of these, namely new_0 and new_1 have encodings that are especially simple: so $\upsilon_*(\text{new}_0)$ and $\upsilon_*(\text{new}_1)$ can be written down in a single pass of the Turing machine M.

Let us now define the block code (ψ_*, ψ^*) used below in the definition of the configuration code (φ_*, φ^*) outlined in Section 3.5:

$$\psi_*(a) = 0^{PadLen}\upsilon_*(a)0^{PadLen}. \tag{4.1}$$

The decoded value $\psi^*(x)$ is obtained by first removing *PadLen* symbols from both ends of x to get x', and then computing $\upsilon^*(x')$. It will be easy to compute the configuration code from ψ_*, once we know what fields there need initialization.

4.2 Rule language

The generalized Turing machines M_k to be defined differ only in the parameter k. We will denote therefore M_k frequently simply by M, and M_{k+1}, simulated by M_k, by M^*. Similarly we will denote the colony size Q_k by Q.

We will describe the transition function $\tau_k = \tau$ mostly in an informal way, as procedures of a program, these descriptions are readily translatable into a set of *rules*. Each rule consists of some (nested) conditional statements, similar to the ones seen in an ordinary program: "**if** *condition* **then** *instruction* **else** *instruction*", where the condition is testing values of some fields of the observed cell-pair, and the instruction can either be elementary, or itself a conditional statement. The elementary instructions are an *assignment* of a value to a field of a cell symbol, or a command to move the head. It will then be possible to write one fixed *interpreter* Turing machine that carries out these rules, assuming that the whole set of rules is a string and each field is also represented as a string.

Assignment of value x to a field y of the state or cell symbol will be denoted by $y \leftarrow x$.

4.3 Fields

In what follows we describe some of the most important fields we will use in the cells; others will be introduced later.

A properly formatted configuration of M splits the tape into blocks of Q consecutive cells called *colonies*. One colony of the tape of the simulating machine represents one cell of the simulated machine. The two colonies that correspond to the current cell-pair of the simulated machine is scanning is called the *base colony-pair*. The formal definition of a colony-pair (as well as some other concepts) must have two forms: one for the program, based on some field values in cells, and one for our reasoning about the history.

Sometimes the left base colony will just be called the *base colony*. Most of the computation proceeds over the base colony-pair. The direction of the simulated head movement, once figured out by the computation, is called the *drift*. The neighbor colonies of the base colony-pair may not be adjacent, in which case the cells will form a *bridge* between them. Possible space between neighbor colonies other the base colony-pair will be filled by stem cells (see below).

Here is a description of some of the fields:

Mode The present behavior of the computation will be characterized by a field of the current cell-pair called the *mode*:

$$Mode \in \{\text{Normal}, \text{Healing}, \text{Rebuilding}, \text{Booting}\}.$$

If its value is Normal we will say that the computation is in *normal* mode. In this case, the machine is engaged in the regular business of simulation. The *healing* mode tries to correct some local fault due to a couple of neighboring bursts of faults (of size β), while the *rebuilding* mode attempts to restore the colony structure on the scale of a couple of colonies. The *booting* mode is used for setting up the structure initially and for starting or continuing the simulation of the Turing machine G of the main theorem.

Info The *Info* track of a colony of M contains the string that encodes the content of the simulated cell of M^*.

Address The field *Addr* of the cell shows the position of the cell in its colony: it takes values in $[0, Q)$.

Drift The direction in $\{-1, 1\}$ in which the simulated head moves will be recorded on the track *Drift*.

Sweep The *Sweep* field keeps track of the number of sweeps that the head made during the work period.

Kind Cells will be designated as belonging to a number of possible *kinds*, signaled by the field *Kind* with values New, Booting, Stem, Member_0, Member_1, Bridge, Outer. Here is a description of their role.

- The kind New has been discussed before.

- A cell is of the Booting kind if it is on the top level of simulation (see Section 2.2).
- Cells of the base colony-pair are of type Member_0 and Member_1 respectively. Members of other colonies have the kind Outer.
- If the two base colonies are close but not adjacent then there will be $< Q$ adjacent cells of type Bridge between them, extending the left base colony towards the right one.
- Stem is the kind of cells filling the space in between colonies other than the two base colonies. They may not have yet been assigned another role, and their previous role may have been erased.

Heal, Rebuild During healing, some special fields of the state and cell are used, treated as subfields of the field *Heal*. In particular, there will be a *Heal*.*Sweep* field. During rebuilding, we will work with subfields of the field *Rebuild*. A cell will be called *marked for rebuilding* if *Rebuild*.*Sweep* $\neq 0$.

Remark 4.5 The Outer kind is redundant: whether a cell is outer can be computed from its *Drift* and *Sweep* fields. But we use it for clarity. ⌟

4.4 Head movement

The global structure of a work period is this:

Simulation phase Compute the new state of the simulated cell-pair, and the simulated direction (called the drift). Then check the "meaningfulness" of the result.

Transfer phase The head moves into the neighbor colony-pair in the simulated head direction called drift (creating and destroying bridges if needed). Redundancy is used to protect this movement from faults.

During the work period the head sweeps, roughly, back-and-forth between the ends of the base colony-pair. The field *Sweep* keeps track of the number of sweeps made within the work period. This together with the address and kind of the current cell is used by the program to determine the actions to be performed.

Definition 4.6 (Front) As the head sweeps in a certain direction, it increases *Sweep* field by 1. The last site in which this increase occurred will be called the *front*. ⌟

Globally in a configuration, due to earlier faults, there may be more than one front, but locally we can talk about "the" front without fear of confusion. The direction of the sweep s is determined by its parity:

$$\text{dir}(s) = (-1)^{s+1}.$$

Bridges between colonies present some extra complication—let us address it.

Definition 4.7 (Gaps) If the bodies of two cells are not adjacent, but are at a distance $< B$ then the space between them is called a *small gap*. We also call a small gap such a space between the bodies

of two colonies. On the other hand, if the distance of the bodies of two colonies is $> B$ but $< QB$ then the space between them is called a *large gap*. ⌟

A large gap between two colonies will be filled by a bridge when they become a base colony-pair. A bridge is always extending the left member colony, except in possibly in the two transfer sweeps while the colony pair is moved. Building a bridge or making repairs may involve "killing" some cells that are in the way and replacing them with new ones, and may involve the replacement action as in Definition 2.9.

Due to the zigging and feathering requirements mentioned in Section 2.4, there will be two kinds of turns: *small* turns and *big* turns. The process uses the parameters

$$Z = \pi^{1+\rho},\ F = Z\pi^{2+\rho} \tag{4.2}$$

with $\rho > 0$. The choices will be motivated in Sections 4.4.2 and 7.3.

4.4.1 Zigging

A *zigzag* movement will be superimposed on sweeping, so that the head can check the consistency of a few cells around the front. The process creates a *frontier zone* of about $2Z$ cells in both directions around the front, where Z was defined in (4.2). This interval is marked by the values of the field

$$ZigAddr \in [-2Z, 2Z)$$

which is undefined elsewhere. Every second step of progress, the head will perform a forward-backward-forward zigzag checking and updating the zone. The turns while doing this are *small turns* defined in Section 4.4.2, a few more steps (normally at most 2) may be needed to find a turning point.

The counting for zigging can be done locally, in the current cell-pair, counting the number of steps needed to move away from the front. For moving back to the front, the counter is not even needed.

Remarks 4.8

1. The need for backward zigging was explained in Example 3.1.
2. The forward-zigging property and the frontier zone will be used in the proof of Lemma 7.15. We will need to show there that in the absence of new noise, after a constant number of passes, a clean interval J of size, say, $\geq 6ZB$ will extend to the right until it reaches the end of a colony-pair or of a rebuilding area. Forward zigging will prevent any disorder from turning the head back prematurely.
3. The choice of Z in (4.2) will be justified in the same proof; it also (generously) lower-bounds the size of new noise capable of turning back the head.

⌟

4.4.2 Feathering

Example 3.2 above suggest that our Turing machine should have the property that between two right-to-left turns on the same point x, it should pass x at least once (similarly for left-to-right turns). We will call this property *feathering*, referring to the picture of the path of the head in a space-time diagram. In fact in some cases we will require more:

Definition 4.9 A Turing machine has the *c-feathering* property for a $c > 0$, if after a right-to-left turn on point x, the next right-to-left turn at a point $\geq x$ must be at a point $\geq x + cB$, and similarly for left-right turns. ⌟

The following example suggests that any computation can be reorganized to accomodate feathering, at the price of a logarithmic factor in time.

Example 4.10 Suppose that, arriving from the left at position 1, the head decides to turn left again. In repeated instances, it can then turn back at the following sequence of positions:

$$1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 5, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, \ldots$$

⌟

If in the original execution the head turned back t consecutive times to the left from position p, then now it will turn back from somewhere in a zone of size $O(\log t)$ to the right of p in each of these times. (Computing the exact turning point is not necessary.)

Our simulating Turing machine will have two different feathering properties: it will obey 1-feathering for all its turns, but on certain kinds of turn called *big turn* will obey F-feathering for the parameter F defined in (4.2).

To remember a turn, a field

$$\mathit{Turned},$$

whose default value is 0, will be set to 1. Suppose that the head arrives at a cell-pair (x, y) from the left. If y has $\mathit{Turned} = 1$ then the head is not allowed to turn left. If it has $\mathit{Turned} = 0$ and the head turns left then y gets $\mathit{Turned} \leftarrow 1$. In both cases, x gets $\mathit{Turned} \leftarrow 0$.

Suppose that the current cell-pair is (x, y), and the next step replaces y by some y' as in Definition 2.9. Then y' inherits the *Turned* value of y.

Analogous rules hold when left and right are exchanged.

The set New of states will only have two elements, new_0 or new_1, where new_i is a state with field $\mathit{Turned} = i$. When a replacement operation takes place as in Definition 7.10, then the *Turned* field of the cell being replaced will be inherited by the cell replacing it.

A "big turn" (see below) will be allowed only in a distance at least F from the just passed place of a previous big turn.

Remark 4.11 As a consequence of the above rule, a cell will *never be killed* just when the head turned back from it. ⌟

Definition 4.12 (Digression) Whenever a turn is postponed since the head is not allowed to turn due to feathering, the simulation to be carried out by the head is suspended until the head returns. This is called a *digression*. ⌟

Definition 4.13 Let

$$PadLen = 4F \log Q. \tag{4.3}$$

For an interval I containing a colony pair we call the the *turn region* the set $\mathrm{Int}(I, -PadLen) \setminus C$, that is the close outside neighborhood of I. ⌟

Small turn Small turns will be done at zigging during normal and healing mode (see later): the head will turn back only at a cell with $Turned = 0$. If one is not found within 3Δ steps (where Δ is defined in (5.1) below) then call alarm but still don't turn. Such an event will be called *small turn starvation*.

In normal mode, a zigging move is done only after every two steps of moving the front: this leaves every second cell with $Turned = 0$ in the wake of this movement.

Big turn Big turns will be done during booting, and the ends of normal and rebuilding sweeps. Consider for example a right-to-left turn; left-to-right turns are similar. It will be attempted at the colony end or the end of a rebuilding area at some position A_0. Due to the organization below and the example above, in all "normal" circumstances, the turn will succeed somewhere in the turn region to the right of A_0, as in Definition 4.13. The front will be carried until to the actual turning place, and then withdrawn back to A_0. The sweep number will increase by 1 only when the head continues left from A_0.

Big turns are governed with the help of the following fields:

$$LeftTurnDist, RightTurnDist, BigDigression, TotalBigDigression.$$

When attempting a big left turn the head moves right towards the planned position A_0. Let the sequence of the previous left turns be $A_1 < A_2 < \cdots$. We can assume $A_0 \le A_1$, though this is not important. Between A_{i-1} and A_i the distance to A_i is in the field *LeftTurnDist*, where it was set after the previous left turn from A_i. Let x be the address of the head. If before reaching the goal we find $x - A_1 < Z$, then the head continues to the right of A_1 to a distance F from A_1, keeping track of this distance in the field *BigDigression* (which was set to 0 at A_1). If before reaching the goal we find $x - A_2 < Z$, then the head continues to the right of A_2 to a distance F, again keeping track of this distance in the field *BigDigression* (reset to 0 at A_2), and so on. There is also a counter called *TotalBigDigression* that is not reset at A_2. If after repetitions of this process, this counter reaches $2Q$ then the big turn attempt has failed: start (or restart) rebuilding, set $TotalBigDigression = 1$ but still don't turn: such an event will be called a *big turn starvation*.

When the planned big turn succeeds then on the way back towards the left, *LeftTurnDist* is set to the distance from the new left turn position, while the variables *BigDigression*, *TotalBigDigression* are reset to 0.

Definition 4.14 The first Z cells on the left of the left turn, with their track *LeftTurnDist* $= 0$ to Z, will be called the *footprint* of the big left turn. Similarly for big right turns. ⌟

Machine M will have the property that after a fault-free path passed over a clean interval, both small turns and big turns can happen without too long digressions. We give here only an informal argument; formal proof must wait until a complete definition of the simulation. Zigs are by the definition spaced by ≥ 2 cells apart, making sure that the points with *Turned* $= 1$ are in general at a distance of ≥ 2 apart. Healing can create only a constant number of segments of *Turned* $= 1$ of size $\leq 3\Delta$ with Δ defined in (5.1). As for big turns, as indicated in Example 4.10 (for $F = 1$), a big turn attempt will be delayed by at most F times the logarithm of the total number of big turns inside a colony or a rebuild area.

The simulated Turing machine will also have the feathering property, therefore the simulation will not turn back from one and the same colony repeatedly, without having passed it in the meantime.

Remark 4.15 The size of the parameter F is motivated by the proof of Lemma 7.15. Here is a sketch of the argument (it can be safely skipped now). At some time t_0 in some interval I we will have clean subintervals $J_k(t_0)$, $k = 1, 2, \ldots$ of size $\geq 6ZB$ in which no fault will appear, and which are separated from each other by areas of size $O(\pi^2 ZB) \ll FB$. For times $t > t_0$ we will track the maximal clean intervals $J_k(t)$ containing the middle of $J_k(t_0)$.

Assume that the head passes over I noiselessly left to right and later also noiselessly from right to left. If the head moves in a zigging way to the right then the Attack Cleaning property will clean out the area between $J_i(t)$ and $J_{i+1}(t)$, joining them. This does not happen only in case of a programmed turn from the right end of $J_i(t)$, in the course of the simulation or rebuilding. But then in the next pair of passes over I, the feathering property implies that the programmed turn from the end of $J_i(t)$ is at least a distance FB to the right. Our choice of F implies that then $J_i(t)$ will be joined to $J_{i+1}(t)$. So two noise-free passes would join all the intervals $J_i(t)$ into a clean area. ⌟

4.5 Simulation phase

The simulation phase, called the `Compute` rule, computes new values for current cell-pair of the simulated machine M^* represented by the current (base) colony-pair, and the direction of the move of the head of M^*. The cell state of M^* will be stored on the track *Info* of the representing colony. The move direction of M^* will be written into the *Drift* field of *each* cell of the base colony-pair (so the whole track must be filled with the same symbol $d \in \{-1, 1\}$.

Let

$$c_{\text{stain}} \tag{4.4}$$

be a constant to be specified later. The rule `Compute` relies on some fixed $(c_{\text{stain}}\beta, 2)$ burst-error-correcting code, moreover it expects each of the words found on the *Info* track to be 2-compliant (Definition 4.1). There is a rule `ComplianceCheck` to check whether a word is 2-compliant.

The rule `Compute` starts with making sure that the input colonies are compliant. Then essentially repeats 3 times the following *stages*: decoding, applying the transition, encoding. It uses some additional tracks like *Work* for most of the computation. The *Info* track will not be modified before all the $Hold[j]$ tracks are written.

In more detail:

1. At the start, the current cell-pair is the left pair of cells of the left member of the base colony-pair. The *Sweep* values of all cells are maximal. Everywhere outside the base colony-pair, the *Drift* values are pointing towards it.
2. During the execution of this rule, the head sweeps the base colony-pair. The big turns for changing the sweep direction will happen in the turn region outside as in Definition 4.13. If there is no neighboring colony then outer adjacent stem cells will be used, or added as needed.
3. For $j = 1, \ldots, 3$, call `ComplianceCheck` on the *Info* track of both colonies of the pair, and write the resulting bit into the $Compliant_j$ track of each.

   Then pass through each colony of the pair and for each address i, if in the cell with this address, the majority of $Compliant_j$, $j = 1, \ldots, 3$ is false, then turn this cell into the ith cell of the colony representing the state new_0. Recall that in Section 4.1, we required that the codes of the states new_i, $i = 0, 1$ are simple enough so that they can be written in a single pass.

   (We could have used any other state instead of new_0 here: just some simple default state is needed.)
4. For $j = 1, \ldots, 3$ do
   a. Calling by $\mathbf{a}$ the pair of strings found on the *Info* track of the interiors $\text{Int}(C, PadLen)$ of the base colonies C, decode it into the pair of strings $\tilde{\mathbf{a}} = v^*(\mathbf{a})$ (the current state of the simulated cell-pair), and store it on some auxiliary track in the base colony-pair. Do this by computing $\tilde{\mathbf{a}} = \text{Decode}(\mathbf{a})$ on some simulated universal Turing machine.

      Each cell stores a track segment of the tape of the machine for Decode. When the head of M is on the cell with the simulated head of Decode—let us call this the *head segment*—then one step of M simulates a number of steps of Decode as large as the size of this segment or until the simulated head leaves the segment (thus changing the head segment).

      For simplicity of analysis, each pair of sweeps changes only the head segment (and possibly the neighbor that becomes the new head segment). We accept this slowdown now.
   b. Compute $(\mathbf{a}', d) = \tau^*(\tilde{\mathbf{a}}, \alpha)$, where α = True if the pair of colonies is adjacent, else False. Since the program of the transition function τ^* is not written explicitly anywhere, this "self-simulation" step is discussed in detail in Section 4.6.

c. Write the encoded new cell states $\upsilon_*(\mathbf{a}')$ onto the $Hold[j].Info$ track of the interior of the base colony-pair. Write d into the $Hold[j].Drift$ field of *each cell* of the left base colony.

   A field called *Replace* is used. Its value can be undefined, or an element of the set New. If one of the new states of the simulated cell pair belongs to New (that is, the rules dictate a replacement, as in Definition 2.9), then write it onto the $Hold[j].Replace$ track everywhere; else the values on the track will be undefined. There is enough capacity in a cell to record this value of a simulated cell (which can have many more states), since the set New has only two possible elements in M^* as well as M.

5. Sweeping through the base colony-pair, at each cell compute the majority of $Hold[j].Info$, $j = 1, \ldots, 3$, and write it into the field *Info*. Proceed similarly, and simultaneously, with *Drift* and *Replace*.

6. If the *Output* field of the simulated cell is defined, write it into the output field of the left end-cell of each colony.

The transfer phase (see Section 4.7) will use the information in the $Hold_j.Replace$ fields.

Part 6 achieves that when the computation finishes on some simulated machine M_k, its output value in cell 0 of M_k will "trickle down" to the output field of cell 0 of M_1, as needed in Theorem 1.

As seen in this construction one can find a constant c_{redund} with the property

$$s_{k+1} \le Q_k s_k \le c_{\text{redund}} s_{k+1} \tag{4.5}$$

(see the definition of s_k in (2.1)), that is a colony of M_k represents a cell of M_{k+1} with redundancy factor c_{redund}. The number of steps U_k taken by M_k needed for one work period of simulation is $O(Q_k^2)$.

Remark 4.16 There is a mechanism more economical on storage, without the full-width *Work* and $Hold[j]$ tracks but with some added complexity, see Section 8.2. This allows a space redundancy factor $1+\delta_k$ with $\prod_k(1+\delta_k) < \infty$, yielding a constant space redundancy factor for the whole hierarchy. ⌟

4.6 Self-simulation

Let us elaborate step 4b of Section 4.5.

4.6.1 New primitives

The simulation phase makes use of the track *Work* mentioned above, and the track

Index

that can store a certain address of a colony.

Recall from Section 4.2 that the program of our machine is a list of nested "**if** *condition* **then** *instruction* **else** *instruction*" statements. As such, it can be represented as a binary string

$$R.$$

If one writes out all details of the construction of the present paper, this string R becomes explicit, an absolute constant. But in the reasoning below, we treat it as a parameter.

Let us provide a couple of *extra primitives* to the rules. First, they have access to the parameter k of machine $M = M_k$, to define the transition function

$$\tau_{R,k}(\mathbf{a}).$$

The other, more important, new primitive is a special instruction

`WriteProgramBit`

in the rules. When called, this instruction makes the assignment $Work \leftarrow R(Index)$. This is the key to self-simulation: *the program has access to its own bits*. If $Index = i$ then it writes $R(i)$ onto the current position of the *Work* track.

4.6.2 Simulating the rules

The structure of all rules is simple enough that they can be read and interpreted by a Turing machine in reasonable time:

Theorem 2 *There is a Turing machine* Interpr *with the property that for all positive integers k, string R that is a sequence of rules, and a pair of bit strings $\mathbf{a} = (a_0, a_1)$ with $a_j \in \Sigma_k$,*

$$\text{Interpr}(R, 0^k, \mathbf{a}) = \tau_{R,k}(\mathbf{a}).$$

The computation on Interpr *takes time $O(|R| \cdot |\mathbf{a}|)$.*

The proof parses and implements the rules in the string R; each of these rules checks and writes a constant number of fields.

Implementing the `WriteProgramBit` instruction is straightforward: Machine Interpr determines the number i represented by the simulated *Index* field, looks up $R(i)$ in R, and writes it into the simulated *Work* field.

There is no circularity in these definitions:

- The instruction `WriteProgramBit` is written *literally* in R in the appropriate place, as "`WriteProgramBit`". The string R is *not part* of the rules (that is of itself).
- On the other hand, the computation in $\text{Interpr}(R, 0^k, \mathbf{a})$ has *explicit* access to the string R as one of the inputs.

Let us show the computation step invoking the "self-simulation" in detail. In the earlier outline, step 4b of Section 4.5 said to compute $\tau^*(\check{\mathbf{a}})$ (for the present discussion, we will just consider computing $\tau^*(\mathbf{a}) = \tau_{k+1}(\mathbf{a})$), where $\tau = \tau_k$, and it is assumed that $\mathbf{a}$ is available on an appropriate auxiliary track. We give more detail now of how to implement this step:

1. Onto the *Work* track, write the string R. To do this, for *Index* running from 1 to $|R|$, execute the instruction `WriteProgramBit` and move right. Now, on the *Work* track, add 0^{k+1} and $\mathbf{a}$. String 0^{k+1} can be written since the parameter k is available. String $\mathbf{a}$ is available on the track where it is stored.
2. Simulate the machine Interpr on track *Work*, computing $\tau_{R,k+1}(\mathbf{a})$.

This implements the forced self-simulation. Note what we achieved:

- On level 1, the transition function $\tau_{R,1}(\mathbf{a})$ is defined completely when the rule string R is given. It has the forced simulation property by definition, and string R is *"hard-wired"* into it in the following way. If $(\mathbf{a}', d) = \tau_{R,1}(\mathbf{a})$, then

$$a_0'.Work \leftarrow R(a_0.Index)$$

  whenever $a_0.Index$ represents a number between 1 and $|R|$, and the values $a_0.Sweep$, $a_0.Addr$ satisfy the conditions under which the instruction `WriteProgramBit` is called in the rules (written in R).
- The forced simulation property of the *simulated* transition function $\tau_{R,k+1}(\cdot)$ is achieved by the above defined computation step—which relies on the forced simulation property of $\tau_{R,k}(\cdot)$.

Remark 4.17 This construction resembles the proof of Kleene's fixed-point theorem, and even more some self-reproducing programs (like a program p in the language C causing the computer to write out the string p). ⌟

4.7 Transfer phase

Before the transfer phase, members of the base colony-pair C_0, C_1 have cells of kind Member_0 and Member_1 correspondingly, with a possible bridge between them. Now the `Transfer` rule takes over: control will be transferred to the neighbor colony-pair in the direction of the simulated head movement which we called the *drift*. During this phase, the range of the head includes the base colony-pair and a neighbor colony in the direction of the drift called *target colony*, including possible bridges between them. At the beginning of the phase, the current cell-pair is the first cell-pair of C_0. Big turns happen in the turn region as in part 2 of the description of the `Compute` rule.

Consider $Drift = 1$. In what follows, whenever the majority of $Hold[j].Replace$, $j = 1, \ldots, 3$ is observed in a cell then we will say that this is a *replacement* situation.

1. Suppose that we don't have a replacement situation. In the first sweep, the head will travel right. Turn all elements of C_0 into outer cells, rebuild the bridge between C_0 and C_1 if necessary, to point

from C_0 to C_1, and turn the elements of C_1 into Member_0 cells. Then continue to the right, start a bridge (if necessary) towards the right (killing all possible non-adjacent stem cells in the way). If the right end of the bridge reaches an outer colony C_2 before Q bridge cells are created, then pass to the right edge of C_2. If Q bridge cells were created, stop at the right edge of this bridge—call it now C_2. Then sweep back to the left end of C_1, while turning the cells of C_2 to kind Member_1.

In both cases, actually go to a distance $3F \log Q$ from the right end of C_2 before attempting to turn. (This way, all later attempted left turns in the next work period will happen to the left of this one, and so any possible cause for alarm is encountered already now, in the transfer process.)

2. In the replacement situation, build a new colony C_1' adjacent on the right to C_0. In the first sweep, perpetuate the value r found on the *Replace* track. Write onto the *Info* track of C_1' the encoding of the value r. (This requires two steps for each created cell x_i of C_0': first it has value $r \in \text{New}$, then it gets address i, and its *Info* field gets the ith symbol of the encoding of r.) Then continue to the end of C_1. On the way back, replace the remaining elements of C_1 with stem cells and set the kind of elements of C_1' to Member_1.

A similar program is executed when $\mathit{Drift} = -1$. The values of *Drift*, *Replace*, *Sweep* and *Addr* always determine what step to perform.

The continuous fault-checking during zigging will notice when a small burst of faults compromises this process (for example when the end of a bridge would "bite" into another colony), by checking whether all boundaries it finds are legal (see the definition of health in Section 5.1) and trigger healing (see later).

4.8 Booting

Ideally, the work of machine M starts from a single active cell-pair of the Booting kind, with addresses $Q-1$ and 0, the middle cell-pair of a yet to be built colony-pair. The *Rider* track of the cell-pair holds a tape segment of the simulated Turing machine G, along with the simulated head. Such a cell-pair will be called a *booting pair*. The segment consisting of this cell-pair will be extended left-right by booting cells, eventually creating a colony-pair, as follows.

Main work Machine M performs Q times the following:

Simulate G in the active pair on the *Rider* track for at most as many steps as the size of the represented tape segment of G, possibly stopping earlier if the represented head would exit that segment earlier. Move the active pair of M left or right as needed, adding cells to the representing segment, and giving them the appropriate addresses of their colonies. All new cells encountered must be stem cells (blanks), and all the ones in the segment already created must be of the Booting kind; otherwise call alarm.

In all this, use zigging and feathering. Zigging uses small turns just as during simulation, but whenever a turn of G is simulated, *make a big turn* (as defined in Section 4.4.2), with the necessary

digressions (see Definition 4.12). When the active pair does not need moving, treat this as (say) a left turn from the point of view of feathering, making the necessary marking and digression. The simulation is continued only after the head returns to the active cell-pair from the digression.

Output If the simulated computation of G terminates (the *Output* field of simulated cell 0 of G is written) then write the same value to the *Output* field of cell 0.

Lifting If the Q steps of M are completed without the termination of the simulated G computation, then create the colony-pair around the original pair of booting cells (and turn its cells into member cells). *Lift* (copy) the *Rider* track of its cells into the *Rider* track of the cell-pair simulated by it. Set the mode of the simulated cell-pair to Booting.

The only error-control during all this is the step counting (the cells are big enough to carry a counter up to Q), zigging, and the call of alarm mentioned in the main work above. This serves to notice when booting cells were introduced by a fault into a non-booting situation. We introduce no mechanism to correct faults during the booting phase, since we do not expect faults during it—see the probability analysis in Section 8.1.

5 Healing and rebuilding

Here we define the part of the simulation program that corrects configurations of machine M that are "almost" healthy.

5.1 Health

Health is that part of structural integrity that can be checked locally, and repaired locally provided it was damaged locally. Structure is maintained with the help of a small number of fields. The required relations among them allow the identification and correction of local damage.

Definition 5.1 The tuple of fields

$$Core = (Kind, Drift, Replace, Addr, Sweep, Rebuild.Sweep)$$

is called the *core*. Recall that a cell is said to *marked for rebuilding* if $Rebuild.Sweep > 0$.

An interval of non-stem adjacent neighbor cells is a *homogenous domain* if its core variables with the exception of *Addr* have the same value, and *Addr* increases left to right. The *left end* of a domain is the left edge of its first cell, and its *right end* is the right edge of its last cell. A *left boundary* the left end of a homogenous domain with either no left neighbor cell or with a neighbor cell belonging to another homogenous domain. Right boundaries are defined similarly. ⌟

Health can be defined formally on the basis of the informal descriptions given here, but the details would be tedious. Recall Definition 4.6 of the front.

- A configuration consists of intervals of non-stem neighbor cells, with possibly stem cells between them. The health for each of these intervals is defined locally. No cell is marked for rebuilding, that is $Rebuild.Sweep = 0$.

  As a non-local condition we will require that exactly one of these intervals contains the front: let us call this the *principal interval*, and that the drift in all other intervals is directed towards the principal one.

- In the principal interval there is a base colony-pair, with possibly a bridge going from one member of the pair to the other one. Let I denote the interval containing this pair. If the sweep inside this colony-pair shows that the `Compute` rule is being performed then the only sweep boundary allowed inside is the front. All other sweep boundaries are outside I, in the turn region as in Definition 4.13.

- During the `Compute` phase, outside the base colony-pair (both in the principal interval and elsewhere), all non-stem cells have their *Drift* value directed towards the base colony-pair. There are possibly colonies of type Outer adjacent to it and each other. They are called *left outer* colonies and their cells left outer cells, or *right outer* colonies depending on whether their drift is $+1$ or -1. Stem cells are also called outer cells (both left and right).

  In part 5 of this phase, the values of *Drift* and *Replace* can change at the front.

- During transfer, the base colony (of the pair) that is in the direction of the drift is possibly extended by a bridge towards the target colony. At this point there is no other bridge, and the front is at the tip of the new bridge.

  In the later part of this sweep, the new bridge already reaches the target colony. If the bridge extends to a full colony then this is converted to the appropriate kinds of member cells in the backward sweep. Domains ahead and behind the front show the changes done. Another possible change occurring at the front is replacement, when dictated by the *Replace* track. In this case the front has the property that, for example when replacing a colony in the right direction, for each member cell created to replace an old member cell, *the address of the new cell is not larger than the address of the one it replaces*.

Definition 5.2 Let us call a bondary *legal* if it can occur in a healthy configuration. ⌟

The following lemma shows that health of an interval of non-stem neighbor cells is locally checkable.

Lemma 5.3 *If an interval of neighbor cells in a tape configuration has only legal boundaries (including those at its ends) then it is healthy.*

Proof. In what follows we don't repeat it but each statement is forced by the kind of boundaries allowed. There are only two basic kinds: a colony boundary and the front, and what matters is the values of the *Core* variables in the cell pair around the boundary.

1. Consider an interval I of neighbor cells. If I contains cells that are not outer, or it contains both left and right outer cells then it also contains the front.
2. Assume that I contains only left outer cells. Then these consist of colonies possibly separated by stem cells, with drift pointing to the right, and possibly a front in a right turn region. The situation is similar when I consists of right outer cells.

   In all other cases I also contains an interval K consisting of neighbor non-outer cells.
3. Let s be the maximum sweep found in K. If s is in the computing phase then K consists of a base colony-pair with a possible inner bridge connecting it. The only possible boundaries inside are a front and the colony boundaries. Turn boundaries are outside it.
4. Suppose now that s is in the transfer phase: then the the drift over K is constant. Suppose that, for example, $Drift = 1$. Look at the description of the transfer phase in Section 4.7. Depending on whether we are in a replacement situation (defined by the *Replace* track) and whether the we are in the first or second sweep, the boundary at the front completely determines the possibilities. The restriction on the addresses mentioned above makes sure that there is enough space for the replacement to succeed.

□

The following lemma is an immediate consequence, given that health is defined by boundaries:

Lemma 5.4 *Let ξ be a tape configuration that is healthy on intervals A_1, A_2 where $A_1 \cap A_2$ contains a whole cell body of ξ. Then ξ is also healthy on $A_1 \cup A_2$.*

Lemma 5.5 *In a healthy tape configuration, over any interval of size $< QB$ there are at most three boundaries between domains.*

Proof. Two colony-ends can be closer than QB to each other in case of a big gap between two neighbor colonies. These and possibly the front can amount to three boundaries. □

In a healthy configuration, the possibilities of finding non-adjacent neighbor cells are limited.

Lemma 5.6 *An interval of size $< Q$ over which the configuration ξ is healthy contains at most two maximal sequences of adjacent non-stem neighbor cells.*

Proof. By definition a healthy configuration consists of intervals covered by full colonies connected possibly by bridges, and possibly stem cells between these intervals. An interval of size $< Q$ contains sequences of adjacent cells from at most two such intervals. □

Lemma 5.7 *In a healthy configuration, a cell's state shows whether it is an outer cell, and also its direction towards the front. The Core track of a homogenous domain can be reconstructed from any of its cells.*

Proof. Whether a cell is outer is computed from some fields. If the cell is outer then *Drift* shows its direction from the front, else the parity of *Sweep* shows it. For booting cells, the number of simulation steps performed increases towards the front. □

5.2 Stitching

We will show that a configuration admissible over an interval of size $> QB/2$ can be locally corrected; moreover, in case the configuration is clean, this correction can be carried out by the machine M itself.

Definition 5.8 (Substantial domains) Let $\xi(A)$ be a tape configuration over an interval A. A homogenous domain of size at least $4c_{\text{stain}}\beta B$ will be called *substantial*. The area between two neighboring maximal substantial domains or between an end of A and the closest substantial domain in A will be called *ambiguous*. It is *terminal* if it contains an end of A. Let

$$\Delta = 24c_{\text{stain}}\beta. \tag{5.1}$$

⌟

Later, in Section 6, we will introduce the notion of *islands*: intervals of size $\leq c_{\text{stain}}\beta B$ with the property that if the configuration is changed in the islands it becomes healthy. Under normal circumstances, there will be at most 3 islands in any interval of size QB. The size of a substantial domain assures that at least one of its cells is outside an island, since even three neighboring islands have a total size $\leq 3c_{\text{stain}}B$.

Lemma 5.9 *In an admissible configuration, each half of a a substantial domain contains at least one cell outside the islands. If an interval of size $\leq QB$ of a tape configuration ξ differs from a healthy tape configuration χ in at most three islands, then the size of each ambiguous area is at most ΔB.*

Proof. The first statement is immediate from the definition of substantial domains. By Lemma 5.5, there are at most 3 boundaries in χ. There are at most 3 islands. Between islands and boundaries there are at most 5 non-substantial domains: of sizes $< 4c_{\text{stain}}\beta B$. The islands have a total size $< 3c_{\text{stain}}\beta B$ and the space between boundaries may add at most $3B$. Adding all these up we get $B(20c_{\text{stain}}\beta + 3c_{\text{stain}}\beta + 3) < 24c_{\text{stain}}\beta B$. □

The following lemma forms the basis of the healing algorithm.

Lemma 5.10 (Stitching) *In an admissible configuration, inside a clean interval, let U, W be two substantial domains separated by an ambiguous area V. It is possible to change the tape on U, V, W using only information in U, W in such a way that the tape configuration over $U \cup V \cup W$ becomes healthy. Moreover, it is possible for a Turing machine to do so gradually, extending, and changing the tape in U or W or enlarging U or W gradually at the expense of V.*

The machine in question can make its turns via "small turns" as defined in Section 4.4.

Proof. At any step below, if we find that $U \cup V \cup W$ is healthy then we stop.

1. If U consists of colony cells then let it be extended towards V until a colony boundary or W is hit. Then do the same with W. If V gets eliminated then the boundary-pair between U and W is necessarily a legal one.

Assume now that the above operations have still left V.

2. If for example U consists of colony cells but W does not, then extend W towards U until V is erased: the boundary obtained must be legal.
3. Assume that both U and V consist of colony cells.

   If both colonies are outer then we can turn all elements of V into stem cells. This situation will not be actually encountered since the front is never near the boundary between two outer colonies.

   If both colonies are inner then turn the cells between them into a bridge from U to V. Continue into it the sweep from U (this is an arbitrary choice).

   If for example U is inner and W is outer then, if the sweep is not the transfer sweep towards W then we fill V with stem cells; else we fill V with bridge cells extending U. In both cases we extend the sweep of U into V.
4. The case remains when neither U nor V consist of colony cells.

   If one of them consists of stem cells then extend it (does not matter which) towards the other until they meet. If both consist of bridge cells then extend either one of them towards the other until they meet. We will always end up with a legal boundary.

□

5.3 The healing procedure

The healing and rebuilding procedures look as if we assumed no noise or disorder. The rules described here, however (as will be proved later), will also clean an area locally under the appropriate conditions.

Healing performs only local repairs of the structure: for a given (locally) admissible configuration, it will attempt to compute a satisfying (locally) healthy configuration. If it fails—having encountered an inadmissible configuration—then the rebuilding procedure is called, which is designed to repair a larger interval.

To protect from noise, any one call of the healing procedure will change only a small part of the tape, essentially one cell: so a noise burst during healing will have limited impact. Every healing operation starts with a survey on a certain interval around its starting point, an illegal boundary called the *center*. If it still finds healing to do then it performs one step of it, and then leaves a trace: adds to an interval of cells around the center marked for rebuilding called a *germ*. (The marking is done on a track different from *Core* and therefore not influencing consistency.) If the germ reaches a size 10β cells rebuilding starts. This prevents rebuilding from starting in cases when healing can succeed. Indeed, a zigging movement in rebuilding will recognize immediately if it has no base of at least 10β rebuilding cells.

Here are the details of healing. Suppose that `Heal` is called at some position. Then it sets *Mode* ← Healing. In what follows, turns will always be small turns, as defined in Section 4.4.2. In any newly created cell, *Drift* is set backwards to the creating cell—to make sure that the head does not get lost on the edge of the infinite vacant space when hit by a burst of faults.

In what follows, when an "attempt" *fails*, increase the germ by 1 cell at one of its ends, go to the center and restart. Survey an interval I consisting of 4Δ cells left and as many right from the center. Let $I' \subset I$ be an interval consisting of Δ cells to the left and as many to the right of the center.

If the ambiguous areas in I cannot be covered by 3 intervals of size $\leq \Delta$ (separated by substantial homogenous intervals) then fail.

If I' is healthy then check the germ. If it is nonempty, decrease it by one cell, go to the center and restart. Else go to the center and turn to normal mode.

If the germ is bigger than 10β cells then start rebuilding.

If I contains a (non-germ) cell marked for rebuilding, remove the mark in the first one (setting *Rebuild.Sweep* ← 0), go to the center and restart.

Find the first illegal boundary x in I'. Attempt to find a substantial domain within Δ steps on the left of x; let S' be the first one.

Attempt to find a substantial domain S'' within Δ steps on the right of x. Let S'' be the first one. If S' and S'' are adjacent (so there is an illegal boundary between them) then fail.

Attempt one step of stitching operation, go to the center and restart.

Note that an ambiguous interval (between S' and S'') does not trigger stitching unless it contains an illegal boundary.

The healing procedure takes at most two sweeps of size 6Δ, so runs in $O(\beta)$ steps.

5.4 Rebuilding

Rebuilding starts by extending a germ (created by healing) to the right, in a zigging way, expecting rebuild-marked cells on the backward zig. This notices if rebuilding started from a burst of faults smaller than a germ in a healthy area, and calls alarm. Just as with healing, we will not mention disorder (since the program does not see it)—but the analysis will take disorder into account.

Rebuilding could also start from big turn starvation (see Section 4.4.2). Since a whole track carries the counter *TotalBigDigression*, this start can also not be triggered by a burst of faults of size βB.

Here is an outline, for rebuilding that started from a cell z on the right end of a germ. The notion of front, of Definition 4.6 can be extended also to the rebuilding mode. We will use the following notion.

Definition 5.11 Suppose that in a configuration ξ, there is an interval I ending in substantial homogenous domains, most 3 ambiguous intervals inside, in which it can be changed to become healthy, with a colony C in $\mathrm{Int}(I, 0.2QB)$. Then we will say that C is a *valid* colony of ξ with a *neighborhood* I. ⌟

The goal is that

r1) we end up with a new decodable area extending at least one colony to the left and one colony to the right of z.

r2) the process does not destroy any valid colony.

The rebuilding operation uses its own addresses, in a special way, to avoid being confused by the occasional deletion or insertion of cells. It has two address tracks: $Rebuild.Addr_{-1}$ counts from the left end of the rebuilding area towards the head, and $Rebuild.Addr_1$ from the right end.

Here are the stages. Recall the stitching operation from Section 5.2.

Mark Starting from the germ, extend a rebuilding area over $4Q$ cells to the right and $4Q$ cells to the left from the center z. Besides the address tracks, use also a track $Rebuild.Sweep$. Every step that changes the configuration must be accompanied by zigging, to check that the rebuilding is indeed going on.

Rebuilding does not assume anything about the content of the area encountered, with two exceptions: right-marking can override left-marking, and there is the possibility of turn starvation as defined in Section 4.4.2.

If leftward marking encounters the front of a rightward marking process that proceeded to a distance of at least *PadLen* as in Definition 4.13, then control is passed to the latter; see Remark 5.12.

The other way that rebuilding can fail in a clean area is turn starvation. Small turn starvation triggers alarm (thus healing). Big turn starvation restarts rebuilding at a distance at least $2QB$ from the supposed turn.

Survey and Create More details of this stage will be given below. It looks for existing valid colonies, and possibly creates some. As a result, we will have one colony called C_{left} on the left of z along with its neighborhood as in Definition 5.11, one called C_{right} on the right of z, and possibly some colonies between them. Make all newly created colonies represent stem cells. Direct all the other colonies with drifts and bridges towards C_{left}. (As during transfer, the creation of a bridge may result also in the creation of a new colony if the bridge becomes Q cells long.) The interval covering C_{left} and C_{right} will be called the *output interval* of rebuilding. The pair of neighbor colonies with C_{left} on its left will be made the current colony-pair.

Mop Remove the rebuild marks (address and sweep), shrinking the rebuilding area R, starting on its left end, onto the right end of C_{left}.

Marked cells from some interrupted rebuilding may remain even after the mop-up operation. These may trigger new healing-rebuilding later.

Details of the Survey and Create stage

The complexity of this stage is due mainly to guaranteeing property (r2) above.

s1) Going from left to right, pass through the marked area, and stitch every pair of consecutive substantial domains separated by an ambiguous area of fewer than Δ cells, just as during healing. But don't create a new germ as in healing: if the stitch result leaves some illegal boundary, just leave it there. The stitching operations may replace some of the germ cells: make sure that every replacement cell is also marked as germ.

s2) Pass through again, and look for whole colonies. Mark the cells belonging to whole colonies as such, and mark all other cells as such. The marks should go to a special track R_1.

s3) Repeat steps (s1) and (s2), writing the resulting marks onto a special track R_2. The following steps will rely on the two tracks R_1, R_2 having identical content. If it is discovered that this is not the case, alarm is called. This is important since the colony creation operations are destructive, they should not be triggered by a single small burst of faults.

s4) Check if there is a marked whole colony to the left of the germ, whose whole neighborhood as in Definition 5.11 is healthy. If yes, find the closest one. If not, create one making sure it does not intersect any marked whole colony, and its neighborhood is healthy (if necessary overwrite part of the neighborhood with stem cells). Call this colony, found or created, C_{left}. Proceed similarly in finding or creating a colony C_{right}.

s5) Fill in the area between C_{left} and C_{right} and the other marked whole colonies between them: fill these gaps with adjacent stem cells, creating a new colony every time an interval of Q adjacent stem cells has been created.

s6) Make all newly created colonies represent stem cells. Let C_0 be the first colony towards the left of the center with at least half of it to the left of the center. Direct all drifts to the left end of C_0. Make C_0 and its right neighbor the new current colony-pair (create a bridge from C_0 to its right neighbor if needed), and let them represent the start of a healing process on the level of M^*.

Remarks 5.12

1. There are only two ways in which the rebuilding process can be interrupted in the absence of noise: overriding from the left (as rightward rebuilding was given precedence), and turn starvation.
2. Once rebuilding completes, since the state of the current colony-pair simulates the start of healing in M^*, the head will continue to the right. There, rebuilding may be called again, but it will not rewrite the colony C_{left}. It may rewrite its right neighbor colony, but not destroy it; so repeated calls to rebuilding will result in progress.
3. Precedence will be used in the proof of Lemma 7.15. There, the goal is to see that in the absence of new noise, after a constant number of passes, a certain clean interval J will extend to the right until it reaches the end of a colony pair or of a rebuilding area. If marking is unstoppable in both

directions then this may not happen soon, since after disorder is entered and exited, no assumption can be made of the state of the current cell-pair. Reaching the left end of J a new rebuilding process may send the head back to the right end, from which a new rebuilding process can send it back to the left end, and so on. If marking to the right has precedence then a new left-directed marking started on the right end would not stop it.

4. Giving precedence to the right rebuilding has the drawback that one can design a initial configuration in which even in the absence of noise, higher-level structure will never arise even locally. Namely, we can fill the line with short intervals $\ldots, J_{-1}, J_0, J_1, \ldots$ each of which is the start (say of size $3Z$) of a right-directed marking process. Then the head, after moving left on J_0, will be captured by the process on J_{-1}, then later captured by the similar process on J_{-2}, and so on. But we will not need to consider such pathological configurations.

⌟

6 Annotation

This section defines the scale-up operation and analyzes trajectories under sparse noise.

6.1 Annotation, scale-up

Let us define the notion of "almost healthy" (admissible) for trajectories. Recall the definition of the parameters Q_k, U_k, T_k in Definition 2.12, and that when for example $T = T_k$ then we denote $T^* = T_{k+1}$, an upper bound on the time of computation of a simulation work period.

Informally, an admissible configuration may differ from a healthy one in a small number of intervals we will call "islands". Even a healthy configuration may contain some intervals called "stains": places in which the *Info* track differs from a codeword. These pose no obstacle to the simulation, and if they are small and few then will be eliminated by it, via the error-correcting code.

Definition 6.1 (Annotation) Consider a region R of space-time where for each time t the set of space-time points belonging to R is an interval $J(t)$. An *annotated trajectory* over R is a tuple

$$(\eta, \chi, \mathcal{I}(\cdot), \mathcal{S}(\cdot)),$$

where η, χ are trajectories over R, the trajectory χ is healthy, further to every interval I of size QB belongs a number $n_I \leq 3$ with the following properties:

a1) At each time there is a set of intervals $\mathcal{I}(t)$ called *islands* and a set of intervals $\mathcal{S}(t)$ called *stains*, where each island is contained in a stain.

The disorder of $J(t)$ in $\eta(\cdot, t)$ is covered by intervals of size $\leq (\beta + 2c_{\text{spill}})B$, one inside each island.

$\eta(\cdot, t)$ differs from $\chi(\cdot, t)$ only in the islands.

a2) In each colony of $\chi(t)$, at any time that does not belong to the last sweep, the *Info* track of the interior can be changed in the stains in such a way that it becomes a codeword of the code υ as in Definition 4.4. There are at most 2 such stains intersecting the interior for each colony.

a3) Stains (and thus also islands) have size $\leq c_{\text{stain}}\beta B$ where c_{stain} was introduced in (4.4).

a4) At any time t, every interval I of size $< QB$ contains at most $n = n_I$ islands. The total size of these islands is at most $\leq (3n+1)\beta B - 2d$, where d is the total size of disorder in them.

If there was no noise during $(t - T^*, t]$, then $n \leq 2$ and every colony contains at most two stains.

a5) In every interval I of size QB, the maximum size of an interval of cells with *Turned* $= 1$, is at most $n_I\Delta$. Also I has at most $3n_I \log Q$ footprints of a big turn closer than $2F$ cells to each other (see Definition 4.14).

a6) Suppose $n_I = 2$ in interval I of size QB to the right of the head. Then one of these islands is in the turn region on the right of a colony C on its left. And either the field *Turned* has value 1 in the cell state represented by C, or the *Drift* track of C has value 1 (in case the represented cell state has already been or is in process of being rewritten). An analogous statement holds when right is replaced with left.

A trajectory η is *admissible* over the space-time region R if it allows an annotation $(\eta, \chi, \dots)$ there. We will say that the history χ *satisfies* η. Its *base colony-pair* at any time t is that of χ (its position is uniquely determined). The head is *free* in an annotation when it is not in any island, and the observed cell-pair is in normal mode.

A configuration ξ is admissible if the one-time-step trajectory consisting of just ξ is admissible. ⌟

When considering a single time t we can refer to $\mathcal{I}(t)$ as $\mathcal{I}$ and $\xi(t) = \eta(\cdot, t)$ as ξ, and we can talk about the annotation of ξ. A configuration may allow several possible annotations; however, since the code defined in Section 4.5 is $(c_{\text{stain}}\beta, 2)$-error-correcting, the codewords recoverable from it do not depend on the choice of the annotation.

Formally, the proof of error-correction will happen by proving that annotation can be extended forward in time; however, we will retain an informal language whenever it is clear how to translate it to annotation. Let us argue (informally, for now), that local correction does not have to deal with more than three islands in any area of size QB.

Example 6.2 (Three islands) Suppose that the head has arrived at some colony-pair C_0, C_1 from the left, goes through a work period and then passes to the right. In this case, if no new noise occurs then we expect that all islands found in C_0, C_1 will be eliminated by the healing procedure. A new island I_1 can be deposited in the last sweep.

Consider the next time (possibly much later), when the head arrives (from the right). If it later continues to the left, then the situation is similar to the above. Island I_1 will be eliminated, but a new one may be deposited. But what if the head arrived to the colony-pair C_1, C_2 and turns back right at the end of the work period? If I_1 is not near the right end of C_0, then the head may never reach it to

eliminate it; moreover, by the feathering way of making turns, it may add a new island I_2 near on the right end of C_0.

When the head returns a third time (possibly much later), from the right, feathering on the level of the simulated machine will cause it to leave on the left. Islands I_1, I_2 will be eliminated but a new island I_3 may be created by a new burst of faults before, after or during the elimination. So the healing procedure must count with possibly three islands possibly in close vicinity to each other. But at least one of these, namely I_2, is near the end of C_0, not in the extended interior $\mathrm{Int}(C_0, PadLen - FB)$. ⌟

Let us now define formally the codes φ_{*k}, Φ^*_k needed for the simulation of history $(\eta^{k+1}, Noise^{(k+1)})$ by history $(\eta^k, Noise^{(k)})$. Omitting the index k we will write φ_*, Φ^*. To compute the configuration encoding φ_* we proceed first as done in Section 3.5, using the code ψ_* there, and then initialize the kind, sweep, drift and address fields appropriately. The value $Noise^*$ is obtained by a residue operation as in Definition 2.4; it remains to define η^*. In the parts of the history that can be locally annotated, and which we will call *clean*, if no colony has its starting point at x at time t, set $\eta^*(x,t) = \mathrm{Vac}$. Otherwise $\eta^*(x,t)$ will be decoded from the *Info* track of this colony, at the beginning of its work period containing time t. More precisely:

Definition 6.3 (Scale-up) Let $(\eta, Noise)$ be a history of M. We define $(\eta^*, Noise^*) = \Phi^*(\eta, Noise)$ as follows. Consider position x at time t, and $J = (t - T^*, t]$. If $[x, x + QB)$ is not contained in some interval I such that η is admissible over $I \times J$ then $\eta^*(x,t) = \mathrm{Bad}^*$. Assume now that it is contained, and let $\chi(\cdot, u)$ be some healthy history satisfying η over $I \times J$. If x is not the start of some colony C in χ then let $\eta^*(x,t) = \mathrm{Vac}$; assume now that it is. Then let $t' \in J$ be the starting time in χ of the work period of C containing t, and let $\eta^*(x,t)$ be the value decoded from $\eta(C, t')$. In more detail, as said at the end of Section 4.1, we apply the decoding ψ^* to the interior of C to obtain $\eta(x,t)$. ⌟

This definition decodes admissible intervals and trajectories for η into histories η^*. (We don't know yet whether these are trajectories of M^*.)

6.2 Isolated bursts

Here, we will prove that the healing procedure indeed deals with isolated bursts of faults. For the elimination of disorder created by faults we will rely on the Escape, Spill Bound and the Attack Cleaning properties of a trajectory in Definition 2.11. Let us give an informal argument first.

Isolated bursts of faults don't create disorder larger than 3β. The head escapes a disorder interval I via the Escape property; while it is inside, the spreading of this interval is limited by the Spill Bound property. Every subsequent time when the head enters and exits I this gets decreased via the Attack Cleaning property, so it disappears after $O(\beta)$ such interactions—see Lemma 6.5 below.

In a clean configuration, whenever healing started with an alarm, the procedure will be brought to its conclusion as long as no new fault occurs. However, as long as the head emerges repeatedly from disorder, we cannot assume anything about the state of the cell-pair to which it arrives. This

complicates the reasoning, having to consider several restarts of the healing procedure. By design, this procedure can change the *Core* track only in one cell. The following lemma limits even this kind of possible damage.

Lemma 6.4 *In the absence of noise, no new island will arise.*

Proof. The islands are defined only by the *Core* track. In normal mode, this track changes only at the front. If this is not the real front, then we are already in or next to an island.

The healing procedure changes the *Core* as part of a stitching operation, or removing or adding a rebuild or germ mark. The proof of Lemma 5.10 shows that inside a healthy area, healing can only change the *Core* track in two ways. Either the front moves left or right, or the *Sweep* values were changed at turn boundaries in the turn region. Neither of these operations affect health.

Adding or removing a rebuild or germ mark affects only the edges of islands. □

The following lemmas are central to the analysis under the condition that bursts of faults are isolated.

Lemma 6.5 (Healing) *In the absence of noise in M^*, the history can be annotated. Also, the decoded history $(\eta^*, Noise^*)$ satisfies the Transition Function property of trajectories (Definition 2.11).*

Proof. In an annotated trajectory, call a space-time point a *distress event* if either a fault occurs there or the head steps onto an island. The extension of annotation is straightforward as long as no distress event is encountered. If after a distress event the head becomes free (according to Definition 6.1), then we will say that *relief* occurred. Let us see what can occur between a distress and relief event.

1. By admissibility, there are at most $n \leq 3$ islands within any interval of size $< QB$, and we will see that this property will be preserved. By the Spill Bound property, the disorder in them can grow to at most $n(\beta + 2c_{\text{spill}})B$. By the Escape property, every time the head is in the disorder it must leave within time $9c_{\text{esc}}(\beta + 2c_{\text{spill}})^2 T$.
2. We will divide the time occurring after a distress event into *stages*: these continue until the head is free at the front, with a complete zigging cycle performed. Stages can be of various kinds:
   - h1) Entering and exiting disorder. We consider here also the case of a burst of noise, which increases n (then by assumption (a4) we had $n < 3$, as the previous burst was long ago).
   - h2) Incomplete call to healing: either not started at its start or hit disorder before completing.
   - h3) Start in rebuilding mode but finish either by hitting disorder or by calling alarm.
   - h4) A complete call to healing.
   - h5) Start in normal mode but meet disorder or alarm before completing a zigging cycle.
3. An island can increase only as a consequence of the head leaving disorder and possibly continuing in case of (h2). It increases by at most 1 in a stage, and due to the Attack Cleaning property, in

each case but the first one per island, this increase is associated by a decrease of the disorder by at least $B/2$. Therefore the bound

$$(3n+1)\beta B - 2d \leq 10\beta B$$

on the total size of islands, required in (a4), is conserved. The germs are included in the islands, so by this bound on their size, rebuilding will not be started by the healing procedure.

4. It follows from the above that the head can leave the disorder at most $(2n+1)(\beta + 2c_{\text{spill}})$ times. This is also a bound on the number of times that case (h1) can occur.
5. Cases (h2) and (h3) imply a case (h1), at their beginning or end, so together they can occur at most $2(2n+1)(\beta + 2c_{\text{spill}})$ times. Each of these stages makes at most two passes over an area of size $< 8\Delta B$, for a total time of $< 32\Delta$ steps.
6. In case (h4) we will either decrease one of the ambiguous areas containing an island, remove a rebuilding or germ mark, or move to case (h5). Since the total possible size of ambiguous areas is $n\Delta B$ and the total size of germs and rebuilding signs is, as seen above, at most $10\beta B$, there are at most $n\Delta + 10\beta$ iterations of the case (h4) that actually change something. The possibility of not changing, just returning to normal mode, can only occur at the end of one of the other possibilities. Therefore case (h4) can occur at most $2n\Delta + 20\beta$ times, and just as incomplete heal, each stage lasts less than 32Δ steps.
7. Case (h5) can end in a case (h1) ($< 7\beta$ times), (h2) (starting at normal, so $< 7\beta$ times) or (h4) (when it actually starts from alarm, so can occur at most $n\Delta + 10\beta$ times). The total number of times is therefore at most $n\Delta + 24\beta$. The head never gets farther than two length of zipping plus $(n+1)\Delta$, that is at most $5Z$ cells, so returning and performing a new zip of at most $8Z$ steps bounds each such stage by $13Z$.
8. Summarizing the estimates: between a distress and relief, there are at most $3(2n+1)(\beta+c_{\text{spill}}) < \Delta$ stages of type (h1) (h2) or (h3) (where we used (5.1)), at most $2n\Delta+20\beta$ of type (h4) and $n\Delta+17\beta$ of type (h5) for a total of at most $(3n+2)\Delta$ stages. Those of kinds (h1)-(h4) take up time $O(\beta^3 T)$, while those of kind (h5) may take up time $O(\beta Z T)$.
9. The extended annotation still satisfies the restrictions on the number of islands, and also the restrictions (a5) on the places with *Turned* $= 1$ and the footprints.

Proof. If at the time of the relief the islands encountered have not been eliminated then there are two possibilities:

- there is one island left, created after the start of the distress.
- the islands left are in a turn region, from which the head turned back in the normal course of simulation.

Indeed, an island can only be left if it is at the bottom of a zig as the front moves forward, since relief requires a whole free zig. Suppose we have the case of the turn region, say on the right. Because of

the restriction (a6) of annotation, at the end of the simulation cycle there is transfer to the right, so the head moves past these islands, leaving at most one.

Contiguous intervals with *Turned* $= 1$ can form only either by a burst of faults, or as the survey interval moves, stage after stage, in the same direction. The latter will only happen as healing is called again and again, eliminating an ambiguous interval gradually by stitching. And since the ambiguous intervals are of size $\leq \Delta B$, it happens at most Δ times.

Footprints of big turns are only created at the edge of colony-pair during simulation. On the right edge, one times $3 \log Q$ such footprints can be created when simulated head moves left, another one when the simulated head returns but moves left again, and a third one when it returns again but will continue right. These cases correspond to $n_I = 1, 2, 3$.

10. The Transition Function property of the history $(\eta^*, \mathit{Noise}^*)$ is satisfied.
Indeed, when the stains are bounded as they are here, the error-correcting code of the simulation program will eliminate them, and computes the transition function correctly. Condition (a5) bounds by a constant the number of steps by which any zigging movements needs to be extended in order to find a place to turn.

New stains only arise in new islands, so they remain bounded again.

□

Lemma 6.6 (Combined heals) *Let $d = 28\Delta B$. Assume that the head moves in a noise-free and clean space-time rectangle $J \times K$ with with $2d < |J| < QB$, $|K| \geq |J|(32\Delta + Z/2)T/B$, touching every cell of J at least once, and never in rebuilding mode. Assume also that at the beginning, J has no interval of $> 3\Delta$ consecutive cells with Turned $= 1$. Then during K, the area $\mathrm{Int}(J, d)$ becomes healthy, with no interval of $> \Delta + 1$ consecutive cells having Turned $= 1$.*

Proof. If healing was not called then after the head touched every cell of J the health of the area is proved. If the head entered J in healing mode then before leaving healing mode, it touches over an area of size $\leq 16\Delta$. The upper bound 3Δ on the number of consecutive cells with *Turned* $= 1$ makes sure that the turns happen within 3Δ cells of both ends of this area, increasing its size to at most $d = (16 + 12)\Delta B$.

Consider some time when healing mode is started while the head is in J, and let I_1 be the interval I defined in the healing procedure for this point. Since rebuilding is not started, this healing succeeds, with the interval I_1' becoming healthy, while staying in an area of size d. After this, new healing starts only at some illegal boundary x outside I_1', and the interval A_1 containing both x and I_1' is healthy at this time. If the head does not leave J during this procedure (and thus $x \in \mathrm{Int}(J, d)$), this healing also succeeds, creating a new healthy interval I_2' that intersects A_1 in an interval of size $\geq \Delta B$. Hence $A_1 \cup I_2'$ becomes healthy. This process continues until the head leaves J. So the only parts of J not becoming healthy are confined to the borders of size d.

As seen in the proof of part 9 of Lemma 6.5, the healing procedure can only create at most Δ consecutive cells with *Turned* $= 1$. Zigging occurs only every 2 steps, so it does not create solid intervals with *Turned* $= 1$.

Let us estimate the time needed for healing J. As in part 6 of Lemma 6.5, each call to healing to shrink an ambiguous interval lasts at most 32Δ steps, for a total of $\leq 32\Delta^2$ steps, for a total of at most $32|J|/\Delta B$.

Let us now count the time needed in normal mode. In this case, the front will move only in one direction. Indeed, suppose that for example that the head entered J on the left, and at some time the front turns in J from right to left. Then the head would travel to a distance $> QB > |J|$ and thus would exit on the left, not reaching the right end of J. In normal mode, every two steps is followed by zigging, so the number of steps in normal mode is at most $|J|Z/2$. □

An interval rewritten by noise can have *Turned* $= 1$ everywhere even if it is clean, so we define a property of intervals avoiding this.

Definition 6.7 A clean interval will be called *safe for turns* if it has no more than $3\Delta + 1$ consecutive cells with *Turned* $= 1$, and has no sequence of more than $3F \log Q$ footprints of a big turn closer than $2F$ cells to each other. ⌟

Lemma 6.8 *If a clean interval is passed over from left to right or right to left without noise then it becomes safe for turns.*

Proof. We will present the proof for a pass from left to right, and point out the only difference for the case when the pass is from right to left. Let $x_1 < x_2 < \cdots < x_n$ be the points of the interval left with *Turned* $= 1$, and let t_i be the times when this happens at x_i.

1. Consider the space-time points (x_i, t_i) where the head makes a big turn in rebuilding mode. Then before time t_i, the head must have performed at least one complete sweep of rebuilding. If this rebuilding succeeds then it leaves a healthy area containing at least one colony on the left and one on the right of its center. Another rebuilding can only start at the left or right of this interval. It cannot be on the left, since t_i was the last time when x_i was passed. So a next rebuilding big turn can only happen about QB cells to the right of x_i.

   If the pass is from left to right then this rebuilding can only fail by big left turn starvation, see Section 4.4.2. This must happen at a distance at least $2QB$ to the intended left turn. If any later rebuilding starts to the left of this, it will already succeed, and we can reason as above.

   In case the pass is from right to left then another possible way that the rebuilding can fail is the leftward rebuilding is overridden by a new rightward rebuilding, see the Marking part of the rebuild procedure in Section 5.4. Since this override can only happen when the front of the rightward process is still at a distance of *PadLen* cells from the leftward front, several of these overrides are at least at a distance of $3F \log Q$ from each other, and therefore the big left turns on the other side will also be at least this far from each other, still guaranteeing safety for turns.

2. Consider the space-time points (x_i, t_i) where the head makes a big turn in normal mode, on the left of a (supposed) colony C. If the simulation finishes normally, and a healthy colony C remains to the right of x_i, then big turns, not belonging to this work period, will only be made at least $\approx QB$ to the right.

   There are two other possibilities. First, C is killed, and (in the same work period) another colony C' is created, overlapping it (possibly even two such colonies, C', C'' if the overlap is very small). Then C' cannot be deleted (without going to another colony on its left), so the big right turns on its left are necessarily separated on the right from others by at least $\approx QB$.

   The other possibility is that rebuilding will be called before the work period over C finishes. This can only happen if then this rebuilding experiences big left turn frustration on its right, since otherwise it would sweep over x_i. Anyway, a new big left turn in normal mode near x_i can only result once, after some rebuilding creates some healthy colony C', and the reasoning continues as above.

3. Consider the points (x_i, t_i) where the head turns in healing mode. If $i < n$ this healing cannot fail, since then the subsequent rebuilding would bring the head to the left of x_i, contradicting the assumption that t_i was the last time when it was there. It follows that the healing will stitch an ambiguous area and the next healing can be started at a distance to the right as large as the size of a substantial area separating it from a next one. So, the size of an interval of cells $x_i, x_{i+1}, \ldots, x_j$ where the head turned in healing mode is at most Δ, the maximum allowed number of cells in an ambiguous area. And there is a gap of the size at least that of a substantial domain between these and the next turning points in healing mode.

   The analysis is similar for points (x_i, t_i) where the head turns in the part of rebuilding when it is attempting to stitch an ambiguous area.

4. What remains is space-time points (x_i, t_i) where the head makes a small turn in normal or rebuilding mode. In these modes the head makes a zig only in every second step, so normally these places also don't occur consecutively.

□

7 Cleaning

This section will scale up the Spill Bound, Escape, Attack Cleaning and Pass Cleaning properties of trajectories, proving them for the history $(\eta^*, Noise^*)$ decoded from a trajectory $(\eta, Noise)$.

7.1 Escape

We will scale up the Escape property in Lemma 7.5 below; here is an outline of the argument. Consider some fault-free path during a time interval J (later we will allow a single burst of faults of size β) over

some space interval G of size $|G| = \lambda QB$. For the times $t \in J$, let $K(t)$ denote the set of those clean points in G that the head passed at least once since they were clean.

The goal is to show that the path will not stay too long in G. This will be since if it stays long then it enlarges $K(t)$, and then builds up colonies in it. These simulate the machine M^*, which commands its head to swing wide according to the program (in zigging or healing), and thus leave G.

Initially, the clean intervals of $K(t)$ can be created using the Pass Cleaning property of trajectories. Every time the head leaves such an interval, it grows via the Attack Cleaning property; so we will mainly be concerned with longer stays. The notion of "long" will be chosen here to make sure that most of it has to be spent in simulating M^*, since both healing and rebuilding finish relatively fast. Let us proceed to details.

Time intervals of length T we may consider as *steps*, since under clean and noiseless conditions, the machine M will perform at least one step of computation during each. Let

$$S = c_{\text{short}} \lambda ZQT, \tag{7.1}$$

with c_{short} a constant to be determined later. We will say that the stay of the head in some maximal interval of $K(t)$ is *short* if it is smaller than S, otherwise it is *long*.

Recall $U_k = U = Q^3$ in Definition 2.12. This is an upper bound on the number of computation steps in one work period, even allowing some calls for healing.

The following lemmas follow the development of a maximal interval $I(t)$ of $K(t)$. An interval I of size QB in $\text{Int}(K(t), 2B)$ is a *manifest colony* if it is healthy with the possible exception of having some rebuild marks (only for survey, not for decision), has undergone a complete simulation work period as part of a colony-pair in a clean subinterval of $K(t)$. In a manifest colony, unless it is at a distance $\leq 2QB$ from the head in the same interval of $K(t)$, the *Drift* track points towards the head.

Lemma 7.1 *The number of manifest colonies does not decrease. Each manifest colony can be followed over time: it may stay in place or shift left or right (without jumping over other manifest colonies).*

Proof. Simulation, or healing does not destroy any part of a colony. It may shift a colony, if the simulation work period of a colony-pair encounters a replacement situation, see Section 4.7. Let us see that rebuilding does not destroy them either.

In the definition of manifest colonies we did allow some (possible leftover) rebuild survey marks, but not decision marks. In order to destroy a colony, the rebuilding process needs to create two decision tracks. One has to consider the case when the rebuilding process is at one end of $I(t)$, hence is fed some uncontrollable information.

Now, after a long stay, the head can exit during a rebuilding process only if it is in its starting stage, marking its interval of operation. Zigging along with attack cleaning implies that in the following short stays between two long ones, before making a decision, the whole rebuilding interval will have to be incorporated into $K(t)$, therefore the decision will be a correct one, not destroying a manifest colony. □

Lemma 7.2 a) *No maximal subinterval of $K(t)$ ever decreases by more than $c_{\text{spill}}B$ on either side.*

b) *The head does not stay longer than $2nUT$ in any subinterval of $K(t)$ of size $\leq nQB$.*

Proof. 1. (a) follows from the No Spill property of trajectories.

2. On (b): If any rebuilding has been started, it will finish in $O(QZ)$ steps unless interrupted. Let us show that it can only be restarted $O(n)$ times while the head stays in the subinterval $I(t)$, hence the total number of steps it can take is $O(nQZ)$.

   There are only two ways that rebuilding can be interrupted: big turn starvation, see Section 4.4.2, or being overridden by the marking process of another rebuilding process on the left. We claim that both of these possibilities can occur at most $O(n)$ times. Big turn starvation occurs only after the head moved to a distance $\geq 4QB$ from the rebuilding center, so the center of a new rebuilding process is at least this far apart. Now consider the override by a rebuilding process on the left. The only reason the overriding rebuilding process was not finished before is turn starvation. Therefore its center is at a distance $\leq 4QB$ to the left. So the number of restarted rebuildings is indeed $O(n)$.

   While no rebuilding starts, the computation is in healing, normal or booting mode. If healing does not start rebuilding then it succeeds, turning to normal mode. New healing can start again, but consecutive healings enlarge the healthy area they are creating; eventually, normal mode computation starts; Subsequent healings can only delay this by a constant factor. The computation in normal (or booting) mode leads to the simulation of cells in M^*. The program of M^*, just like that of M, proceeds by sweeps (zigging or healing). Even the shortest of these sweeps has size at least $2\beta QB > \lambda QB$, therefore a full sweep will exit $I(t)$ in $\leq nUT$ steps.

   If rebuilding succeeds, it creates a colony-pair that simulates a cell-pair at the start of healing mode, hence starting a sweep of size $> \beta QB$ to the right. It may be interrupted by rebuilding again, but this rebuilding results in a new colony-pair of the same kind, in $O(QZ)$ steps, at the place where it started, at least QB to the right of the last working colony-pair. So exit happens again in $\leq nUT$ steps.

□

Lemma 7.3 *Suppose that during a long stay in $K(t)$ a rebuilding process is completed. It result is a pair of neighbor manifest colonies, on the left and right of the center from which rebuilding started. Let us call these the* left result *and* right result *of rebuilding.*

A manifest colony can only become a left result once, and a right result once.

Proof. The smallest interval D containing the resulting colony-pair will be healthy and safe for turns at the time when rebuilding finishes. The following development will never introduce inconsistency into D, other than rebuild marks resulting from some rebuilding process started outside it.

The only way in which a rebuilding process can start even in a healthy area is big turn starvation, as defined in Section 4.4.2. But in the present case, the rebuild process looking for a big turn must have started looking for a turn outside D, and since D is safe for turns, it would have found a turning

point close to an end of D, so the left colony of D could not become its left result, nor the right colony of D its right result. □

Lemma 7.4 *The number of long stays is at most* 3λ.

Proof. 1. Consider some maximal interval $I(t)$ of $K(t)$, and a long stay in it.

Suppose first that no rebuilding process is triggered or continued during the stay; then only healing and computation steps are possible.

Suppose that there was no manifest colony in $I(t)$ before entry. The stay is long enough that at least one complete work period will be performed on a neighbor colony-pair. So by the time the head leaves, there will be at least one manifest colony; in fact the exit will happen during a transfer process from a manifest colony (possibly slowed down by healing).

In general, whenever the exit happens after a long stay then it either happens this way or during the marking stage of a rebuilding process.

Suppose there were manifest colonies at entry time, and the head enters on the left. Let C_0 be the leftmost manifest colony of $I(t)$. The head can pass to C_0 only as a consequence of a transfer process from some colony C_{-1}. So the long stay either adds C_{-1} as a new manifest colony, or joins $I(t)$ with another subinterval of $K(t)$ on its left, which contains a manifest colony C_{-1}.

2. Suppose now that a rebuilding process starts or continues during the long stay. Then it either terminates, will be overridden by another rebuilding process, or triggers another one via turn starvation. This can happen repeatedly, but each of these processes has a center of at least QB from the previous one, so one of them has to succeed since otherwise the stay would not be long. When it terminates, it creates a resulting colony-pair.
3. We found that each long stay either adds a new manifest colony, or joins two subintervals of $K(t)$ of size $\geq QB$, or creates a new left result and a new right result. Since there are at most λ manifest colonies in $K(t)$, there can be at most λ creation events, and $\lambda - 1$ events of joining two disjoint subintervals of $K(t)$ of size $\geq QB$. Hence the total number of long stays of kind 1 is at most $2\lambda - 1$, and the total number of long stays of kind 2 is also at most λ.

□

The following lemma is the scale-up of the Escape condition.

Lemma 7.5 (Escape) *Let* c_{esc} *be as defined in* (2.5). *In the absence of Noise***, the head will leave any interval* G *of size* nB *with* $n = \lambda Q$, $1 \leq \lambda \leq 3\beta$, *within* $c_{\text{esc}}\lambda^2 U$ *steps.*

Proof. Consider a time interval of length $c_{\text{esc}}\lambda^2 UT$. Suppose that a burst of faults of size $\leq \beta$ happens during it: then we will consider the larger part J of the time interval before or after the burst (or the whole interval if there is no fault). Let

$$d = c_{\text{marg}}, \ g = c_{\text{esc}}(7d)^2.$$

Let t_0 be our starting time. Subdivide the interval G into subintervals L_i of size dB called *blocks*. The numbering of L_i should be such that head at time t_0 is in block L_0, so i can have negative values. Let the blocks with $i \equiv 0 \pmod 4$ be called *boundary* blocks, and those with $i \equiv 2 \pmod 4$ called *middle* blocks. Consider the interval of adjacent blocks $I = L_{i-3} \cup L_{i-3} \cup \cdots \cup L_{i+3}$. If the head is in L_i then by the Escape property of trajectories and (2.6), it will escape I within time gT. If it is at time t_j in L_i then let t_{j+1} be defined as the time at which it leaves I. This defines a sequence of times t_j. The time intervals $(t_i, t_{i+1}]$, all of size $\leq gT$, will be called *skips*. The skip is either to the left of I or to the right: if to the left, then its *middle block* is L_{i-2}, if it is to the right then L_{i+2}.

1. The number of skips in which the middle block is not clean is at most $\pi n/2d$.
   Proof. The total number of middle blocks is $\leq n/4d$. If a block is the middle block of a right skip π times, then the Pass Cleaning property implies that it becomes clean. Similarly for left skips. So it can be middle block for only 2π skips.
2. The number of skips during which the head touches disorder is at most $\pi n/2d + 6n/d$.
   Proof. We already estimated the number of skips whose middle block is not clean. The remaining skips pass over a clean middle block, but may touch disorder before or after it. If they do this then they will have to either enter a clean middle block from disorder, or leave it. By the Attack Cleaning property, each leaving skip increases the clean interval it leaves by $\geq B/2$. There are only 6 other blocks in the range of the skip, with a total length $6dB$, so after 12 leaving skips, they would be cleaned. The entering skips will have to be balanced by leaving skips, so the total number of skips touching disorder with a given middle block is ≤ 24. There are at most $n/4d$ middle blocks, bounding the total number of these skips by $6n/d$.

Now consider the skips in which the head does not touch disorder (clean skips).

3. The total number of short stays containing clean skips is at most $6n/d$.
   Proof. Each short stay ends with a skip that touches disorder.
4. Let us add up all the estimates.
   Part 2 shows that the number of skips that are not clean is at most $\pi n/2d + 6n/d$, for a total time, with $n = \lambda Q$.

   Part 3 shows that the number of short stays containing clean skips is at most $6n/d$, for a total time of at most $6\lambda QS/d$ where S was defined in (7.1).

   Lemma 7.4 bounds the number of long stays by 3λ, and Lemma 7.2 bounds the length of each long stay by $2\lambda UT$, so the total time taken by long stays is at most $6\lambda^2 UT$. Given that $U = Q^3$, this last term dominates the two previous one, so for large Q the sum will be bounded by $7\lambda^2 UT$. Since (2.5) defined $c_{\text{esc}} = 7$, this completes the proof.

□

7.2 Weak attack cleaning

This section will scale up the Attack Cleaning property of trajectories (Definition 2.11) to machine M^*, but first only in a weaker version, restricting the number of bursts of faults in the relevant interval.

Definition 7.6 (Trap) Recall the definition of trains in Section 4.4.2. In an interval I, clean except possibly up to 3π islands of size β, a point is considered a *leftward trap* if (after changing it in the islands), it is at a right-directed frontier zone with a footprint of a big right turn (as in Definition 4.14). ⌟

The Attack Cleaning property says the following for the present case. Let P be a path that is free of any fault of η^*. For current colony-pair (x, x') (where $x' < x + 2QB$), suppose that the interval $I = \left[x - c_{\text{marg}}QB, x' + QB\right)$ is clean for M^*. Suppose further that the transition function, applied to $\eta^*(x,t)$, directs the head right. Then by the time the head comes back to $x - c_{\text{marg}}QB$, the right end of the interval clean in M^* containing x advances to the right by at least $QB/2$.

Lemma 7.7 (Weak attack cleaning) *In addition of the above condition of attack cleaning, assume that the faults of size of trajectory η in the interval I covered by at most $s < Q/3E$ bursts of size β. Then the conclusion holds.*

Proof. The computation phase of the simulation on the colony-pair (x, x') is completed, then the transfer phase begins, possibly entering the disorder to the right of $x' + QB$. We argue that there are only two ways for the head to get back to $x - c_{\text{marg}}QB$.

at1) The transfer into a new colony-pair with starting point $y \geq x' + QB$ succeeds despite the disorder, and the clean interval extends over it, before the head moves left to $x - c_{\text{marg}}QB$ in the course of the regular simulation. Some inconsistencies may be discovered along the way, but they are corrected by healing.

at2) The inconsistencies encountered along the way trigger some rebuilding processes. Eventually a complete, clean rebuilding area is created, the rebuilding succeeds, leaving a clean colony also to the right of $x' + QB/2$.

By the Spill Bound property, disorder can spill left of $x' + QB$ only by $c_{\text{spill}}B$ as long as the path P is fault-free. If a burst of faults creates an island at a distance $\geq EB$ from the disorder and other islands then this island will be healed unless it is at a rightward trap. The feathering property assures that such remaining islands are at a distance $\geq ZB/3$ from each other, ready to be healed at any next pass. There are at most s bursts of size $\leq \beta$, so the possibly not correctable islands are all within distance sEB of the disorder, to the right of

$$R = x' + QB - (sE + c_{\text{spill}})B.$$

Consider this as the new left end of the disorder; the bound s on the number of bursts implies that size of the new spill is still $< QB/3$.

Suppose that rebuilding is not initiated (with creating a substantial germ): then the head can move deeper left into the colony of x' only by the normal course of simulation: the transfer stage of the simulation must be carried out, and this requires at least as many attacks to the right as the number of sweeps in the transfer stage. Every attack (followed by return) extends the clean interval further, until the whole target colony becomes clean, and the transfer completed. This is the case (at1).

Recall the rebuilding procedure in Section 5.4: the rebuilding area extends $3Q$ cells to the left and right from its initiating cell. This may become as large as $6QB$ to the left and right. If initiated, its starting position z is to the right of R as defined above. It then may extend to the left to at most $z - 6QB$ (this is over-counting, since the cells of the colony of x are all adjacent): if the head moves to the left of this, then the rebuilding must have succeeded. Its many sweeps will result in attacks that clean an area to the right of the starting point. The procedure may be restarted several times, but those re-startings will also be initiated to the right of z. The rebuilding also must find or create a colony manifestly to the right of the restarting site. Since $R > x' + QB/2$ this will move the boundary of the area clean in M^* by at least $QB/2$: this is the case of (at2).

In the process described above, it is possible that the rebuilding finds a competing colony C starting at some $y \in x' + QB + [-\beta B, 0)$ which slightly (by the size of an island) overlaps from the right with the colony of x. The rebuilding may decide to keep C and to overwrite the rest of the colony of x' as a bridge (or even target). This does not affect the result. □

The following lemmas draw the consequences of several applications of weak attack cleaning.

Lemma 7.8 *Let P be a path with at most $Q/2E$ bursts of faults of size β over an interval $I = [a, b)$ that is admissible at the beginning of P. Then by the end of P, the subinterval* $\mathrm{Int}(I, 13QB)$ *will be still admissible. If no bursts occur on sections of the path that enter and exit I from the right then the subinterval $[a + 13QB, b)$ will also stay admissible.*

Proof. Let us divide the bursts of faults into three groups. As bursts intersecting I occur, let us call each island created by a new burst an *end-island* if it is closer than $2EB$ to either one of the ends of I or to an earlier end-island. Let us call the other islands within $7QB$ of the ends the *rebuilding* islands, and the rest the *interior* islands.

By definition the end-islands are confined to within QB of the ends of I. If the head enters into any colony away from the end-islands in normal mode, then it will correct any two islands that are within $2EB$ from each other (an isolated one that was there and a new one just created). A burst can create an island that is not corrected within one pair of sweeps of the simulation, only at the end of a big turn. These big turns are outside the interior of any extended colony and separated from each other by at least $ZB/3$, therefore the islands at their ends will be corrected in any later pass by the head that touches them.

Bursts can also affect rebuilding, but again the ones that do not become end-islands and are not corrected in one sweep are only the ones at the end of big turns. Even repeated rebuilding does not allow the remaining islands to be closer than $ZB/3$ to each other, therefore any new rebuilding process

that touches one of these will correct it. It follows that if a rebuilding process is started at a distance of least $7QB$ from the edges then, since it does not reach any end-island, it will be able to complete, not reaching the interior $\mathrm{Int}(I, 13QB)$. Therefore the head will always reach this interior in normal mode.

The argument clearly proves also the last statement of the lemma, about a path with no faults on entering from the right. In this case there is at most one end-island on the right end. Indeed, if the path returns from the right end-colony, leaving an island there, then due to feathering by the simulated trajectory, next time it must move right from it, and by passing the island, must correct it. □

Lemma 7.9 *Let I_0 be an interval of size $> 28QB$, and J an adjacent interval of size kQB on its right. Assume that I_0 is super-healthy at the beginning of a path P whose faults are coverable by fewer than $Q/2E$ bursts over the interval $I_0 \cup J$, that passes I_0 at least 2^{k+14} times from left to right. Assume also that no faults occur on segments of the path that enter and exit I_0 from the right. Then at some time during the path, the interval J becomes admissible. The same statement holds if we switch left and right.*

Proof. We will apply Lemma 7.8 to intervals into which the I_0 and the right extensions of its admissible area. It is easy to see that $\mathrm{Int}(I, 13QB)$ will always stay admissible; at any time t let $I(t)$ be the largest admissible interval containing this. At the first pass, the rightmost colony of $I(t)$ will be at a distance $\leq 13QB$ from the right end. The lemma implies that the right end of $I(t)$ will not move left. Every time when P passes to the right of $I(t)$, there will be an attack, extending $I(t)$ by QB. However, not every time that path P passes I_0 to the right, will it necessarily pass also $I(t)$; it may turn back before. Feathering makes sure, however, that in 2^i passes, it moves right at least i times, so the 2^{k+14} passes are sufficient to include extend $I(t)$ over the whole interval J. □

7.3 Pass cleaning

We will assume that π is an even number. The scaled-up version of the Pass Cleaning property considers a path P with no faults of η^*, as it makes

$$\pi^* = \pi + 8 \tag{7.2}$$

passes over an the interval I of size $c_{\mathrm{pass}}QB$, and claims that they make $\mathrm{Int}(I, c_{\mathrm{marg}}QB)$ clean for η^*. The weak version of this property uses the additional assumption that the faults of P are coverable by at most 3π bursts of size β. This is what we will prove first, so let us use this assumption. Since there are few bursts, there will be long intervals that are fault-free. Specifically, I is made up of clean subintervals of size $\geq 4ZB$ that we will call *basic holes* separated from each other and the ends by distances $\leq 12\pi ZB$.

The pass cleaning property of η cleans the basic holes (except for margins of size $\leq c_{\mathrm{marg}}B$). By the Spill Bound property, they may erode on the edges by a further amount $c_{\mathrm{spill}}B$. We will call these

somewhat smaller intervals still basic holes. One more pass will make the basic holes, according to Lemma 6.8, safe for turns. The following two lemmas will show how some order will be established on them in two more passes. Recall that the maximum number of cells in a healing area, $E = O(\beta)$ from Definition 5.8, is a constant, much smaller than the zigging distance (in cell widths) defined in (4.2) as $Z = \pi^{1+\rho}$.

Definition 7.10 A clean interval J of size $> 3ZB$ not containing the head is called *right-directed* if it is safe for turns, the head is to the right of J, and its cells, maybe with the exception areas of size $3\pi EB$ on the ends, point towards a front at the right end of J (in normal operation or rebuilding), with the corresponding frontier zone inside J. ⌟

Lemma 7.11 *Consider an interval $[a, b)$ of size $\geq 4ZB$ that is safe for turns. If a path passes it noiselessly from left to right then it will leave a clean right-directed interval $[a, b')$ with $b' \geq b - c_{\text{spill}}B$. The same is true when interchanging left and right.*

Proof. Assume that the starting mode is normal. If it remains normal then J naturally becomes directed by the time the head exits. Since it exits at the front (more precisely at the right of the frontier zone), the frontier zone stays behind. Let us see that also in all other cases when the head exits it leaves behind the right frontier zone.

If healing gets triggered then as long as healing succeeds it extends a healthy area. It moves towards right only if the front is on the right. If it does not succeed then rebuilding gets triggered, and since it does not touch the left end anymore, it either succeeds or exits on the right while trying to extend towards the right. The same considerations work if the starting mode is healing, except possibly on the part of size $\leq EB$ of the left end where the first healing took place, whose purview was not completely contained in J. □

Recall the definition of the feathering parameter F in (4.2).

Lemma 7.12 *Consider a fault-free path.*

a) *Suppose that an interval J is right-directed, and the head enters it from the right. If the head leaves on the right then J will stay right-directed.*

b) *If the head leaves on the left then within $2Z$ cells of the right end of J there will be a footprint of a big left turn in J (as defined in Section 4.4.2).*

The same conclusions hold if we switch left to right.

Proof. To (a): Both in normal operation and rebuilding, the head moves the front with itself and returns to it from every zig, except when it is captured at an end of J (by faults or disorder). If an island is encountered or faults occur then this will be healed, except when the healing interval (whose maximum size is EB) intersects the boundary of J (where the disorder may capture the head).

To (b): the front can turn back only at leftward traps; otherwise the simulation or rebuilding will move it forward, and the forward zigging prevents faults or disorder from turning it back prematurely.

On turning back, it will leave behind a footprint of a big left turn, though the at most 3π bursts of size $\leq \beta$ covering the faults happening in between can shorten this train (of size $Z/2$) by $3\pi \ll Z$ cells.

□

Lemma 7.13 (Weak pass cleaning) *Suppose that a path P has no faults of η^*, it makes π^* passes over the interval I starting from the left, with its faults covered by at most*

$$s = \pi^*(\pi^* + 2^{c_{\text{pass}}+c_{\text{marg}}+14}).$$

bursts of size β in I. Then by end of the π^*th pass the interior $\text{Int}(I, c_{\text{marg}}QB)$ *becomes clean for η^*.*

Proof. Let us number the passes after the first π of them like pass 1,2,3,....

1. The first $\pi + 1$ passes make the basic holes clean and safe for turns, except for margins of size $\leq (c_{\text{marg}} + c_{\text{spill}})B$.

   Proof. As shown above, the basic holes, of size $\geq 4ZB$ and separated from each other and the ends by distances $\leq 4sZB$, become and stay clean for η in the first π passes, except for margins of size $\leq (c_{\text{marg}} + c_{\text{spill}})B$. Now the next pass, called pass 1, will make the basic holes safe for turns.

Assume that pass 1 was from right to left; otherwise, we start one pass later. Since we will finish in 7 passes we will still finish by $\pi^* = \pi + 8$. Let us call at any time a subinterval of I a *hole* if it is clean and safe for turns with the possible exception of s islands, and is maximal with this property. Holes will grow from basic holes.

2. Passes 2 and 3 will leave each hole left-directed, with the footprint of a left turn on the left end of each but possibly the last one.

   Proof.

   - Lemma 7.11 shows that the next pass turns each basic hole into a right-directed hole, with possibly a single island caused by faults.
   - Lemma 7.12(a) shows that the limited number of bursts of size β between this and the next pass that cover the faults, keeps the holes right-directed (just possibly adding some islands).
   - Lemma 7.11 shows again that the following leftward pass turns each hole into a left-directed one. Lemma 7.12(b) shows that this same leftward pass leaves a footprint of a left turn within EB of the right end of each hole.

3. After pass 4 (which is from the left), each interval between holes has a footprint of a big left turn on its left and one of a big right turn on its right.

   Proof. Feathering allows the footprint of a big left turn on the right end of a hole J to be erased only if it is moved to a distance at least $\geq FB$ to he right. As $F \gg$ the upper bound sZ on the separation between the holes, the process erases the disorder between J and the next one, J' unless J' contains a rightward trap at its end. In the latter case, a footprint of the big left turn at the right end of J remains. Lemma 7.12(b) shows that a footprint of a big right turn is created on the left end of every remaining hole.

4. Passes 5 and 6 make the interval clean and safe for turns, except possibly one island of size β caused by faults during pass 6.

   Proof. Having each boundary between holes surrounded by the footprints of big left and right turns, the next pass (which is from the right) will erase all these boundaries. So it makes the whole interval (other than the edges) clean. The following pass will make it safe for turns.

5. Pass 7 will clean $\mathrm{Int}(I, 7QB)$ for η^*.

   Proof. The intrusions from right to left into I before the next pass may create up to s islands of disorder, of size β, but these islands are placed at least $ZB/3$ apart. Indeed, in order for an island not to be cleaned, it must be at the right end of a big left turn, and so will leave a left turning footprint. So a later intrusion can only leave an island if it does not see $2E$ cells of this footprint and is therefore at a distance $> ZB$ to its left, or passes it by a distance FB. These intrusions create no rightward trap.

   When the full rightward pass comes, the head cannot pass without either already having an area already clean for η^* or cleaning, healing and rebuilding. No rebuilding started farther than $6BQ$ from the boundary will exit I: it may only result in alarm if it encounters disorder or the single burst of faults of size $\leq \beta$ occurs. As seen above, any disorder encountered at a time will be of size $\leq \beta B$, at a distance at least $ZB/3$ from the next one. Hence once the healing restarts after this disorder is cleaned, it will discover the rebuilding front, allowing it to continue. So eventually the rebuilding succeeds.

   This process may be repeated as the head moves right, and indeed the head cannot pass through I without continuing this process, making it super-healthy. The only missed areas are those that may contain a rebuilding interval reaching the edge of I.

   □

Lemma 7.14 *Consider the statement of Lemma 7.9 with the interval I_0 having the same length kQB with $k = c_{\mathrm{pass}} + c_{\mathrm{marg}}$ as the interval J. The conclusion holds also without the bound on the number of bursts covering the faults over the interval $I_0 \cup J$.*

Proof. Let $J_1 = I_0$, $J_0 = J$. We will show that if the conclusion does not hold then the paths passes over all the infinite sequence of consecutive adjacent intervals $J_2, J_3, \ldots$ on the left of J_1, of size $|J_1|$. Since the path is finite, this leads to a contradiction. Let $i = 1$.

1. Suppose the conclusion of Lemma 7.9 does not hold. Then the faults needed at least $Q/2E$ bursts of size β to cover them over $J_{i+1} \cup J_i$ during this time, consequently at least $\pi^* + 2^{k+14}$ bursts happened during some two consecutive pair of the 2^{k+14} rightward passes over J_{i+1}.
2. By the Escape property, each fault is covered in a burst of size β contained in a segment covering an interval $> 3c_{\mathrm{pass}}QB$ with no more faults outside the burst. Since these segments don't pass over J_{i+1}, each contains a fault-free pass over J_{i+2}.

Suppose that $\mathrm{Int}(J_{i+2}, c_{\mathrm{marg}}QB)$ becomes clean at some time during the first π^* of these passes. There are still 2^{k+14} fault-free passes over J_{i+2}, so we are back at the situation of part 1 with $i \leftarrow i+1$.

3. Suppose that J_{i+2} does not become clean during the first π^* passes. Then by Lemma 7.13, since $k = c_{\mathrm{pass}} + c_{\mathrm{marg}}$, the number set of faults need a number bursts of size β covering them in J_{i+2} exceeding $s = \pi^*(\pi^* + 2^{k+14})$. Then at least $\pi^* + 2^{k+14}$ bursts happen over J_{i+2} between some consecutive pair of the left-right passes over J_{i+2}. By the Escape property, each burst is contained in a segment covering an interval $> 3c_{\mathrm{pass}}QB$ with no more fault outside the bursts. Since these segments don't pass over J_{i+2}, each contains a fault-free pass over J_{i+3}. This brings us back to the situation of part 2 with $i \leftarrow i+1$.

□

Let us remove the bound on the number of bursts in the Pass Cleaning property of η^*

Lemma 7.15 (Pass cleaning) *Let P be a space-time path without faults of η^* that makes at least π^* passes over an interval I of size $c_{\mathrm{pass}}QB$. Then there is a time during P when $\mathrm{Int}(I, c_{\mathrm{marg}}QB)$ becomes clean.*

Proof. Let $J_1 = I$. We will show that if the conclusion does not hold then the paths passes over all the infinite sequence of consecutive adjacent intervals $J_2, J_3, \ldots$ on the left of J_1, of size $|J_1|$. Since the path is finite, this leads to a contradiction. Let $i = 1$, $k = c_{\mathrm{pass}} + c_{\mathrm{marg}}$.

1. By weak pass cleaning (Lemma 7.13), if $\mathrm{Int}(J_i, c_{\mathrm{marg}}QB)$ did not become clean for η^*, the number of bursts of size β in J_i needed to cover the faults is more than $s = \pi^*(\pi^* + 2^{k+14})$. Therefore there is a time interval between two consecutive left-right passes over J_i with at least $2(\pi^* + 2^{k+14})$ bursts over J_i. Due to the Escape property, each part of P containing one of these bursts has a noise-free segment of size $> 2c_{\mathrm{pass}}QB$ either on the left or on the right of J_i. Without loss of generality we can assume that at least half of them are on the left, giving $\pi^* + 2^{k+14}$ noise-free passes over J_{i+1} during this time.
2. If J_{i+1} does not become clean during the first π^* of these passes then restart the reasoning, going back to part 1, setting $i \leftarrow i+1$. Otherwise by Lemma 7.14, interval J_i becomes clean during the next 2^{k+14} noise-free passes over J_{i+1}, contrary to the assumption.

□

7.4 Attack cleaning and spill bound

Let us remove the bound on the number of bursts in the scale-up of the Attack Cleaning property.

Lemma 7.16 (Attack cleaning) *Consider the situation of Lemma 7.7. The conclusion holds also if the number of bursts of size $\leq \beta$ needed to cover the faults of η in $I = \left[x - c_{\mathrm{marg}}QB, x' + QB\right)$ is not bounded by $Q/3E$.*

Proof. By Lemma 7.7, we need now only to consider the case when there are at least $Q/3E$ bursts. Consider the path $P' \subseteq P$ containing the first $\pi^* + 2^{c_{\text{marg}}+16}$ of these. The Escape property implies that P' passes the interval J of length $c_{\text{pass}}QB$ on the right of I this many times. The Pass Cleaning property then implies that $\text{Int}(J, c_{\text{marg}}QB)$ becomes clean for η^* during P'. Then Lemma 7.14 (applied in the left direction) implies that within the next $2^{c_{\text{marg}}+16}$ passes, the disorder of η^* of length $\leq (1+c_{\text{marg}})QB$ between the old clean interval ending at $x' + QB$ and the new one beginning at $x' + (c_{\text{marg}} + 1)QB$ will be erased. □

Let us prove the scaled-up version of the spill bound property.

Lemma 7.17 (Spill bound) *Suppose that an interval I of size $> 2c_{\text{spill}}QB$ is clean for η^*. and let P be a path that has no faults of η^*. Then $\text{Int}(I, c_{\text{spill}}QB)$ stays clean for η^*.*

Proof. Without loss of generality, consider exits and entries of the path on the left of I. Let C_0, C_1 be the two leftmost colonies in I, where by definition C_0 is at the very end of I. We can assume that C_0 is damaged since the Spill Bound property of η allows a spill of size $c_{\text{spill}}B$ into it. Consider a part of the path entering I and then leaving again on the left. As long as disorder is at least at a distance EB away from C_1, when the path enters these colonies it just continues the simulation. Any burst of faults will be corrected or leave an island subject to the limitations of health.

The islands created by faults in C_0 can be divided into two groups: one is a group starting from the left end in such a way that the distance between consecutive elements is at most EB.

The other ones are at a distance $\geq ZB/3$ from each other, at left turning footprints. It follows that if the faults can be covered by at most $Q/3E$ bursts of size β in P over C_0 then no colony on the right of C_0 will be damaged, since the first group never passes beyond C_0. On the other hand, if than $Q/3E$ bursts are needed in C_0 then we can finish just is in the proof of Lemma 7.16. □

8 Proof of the theorem

Above, we constructed a sequence of generalized Turing machines $M_1, M_2 \ldots$ with cell sizes $B_1, B_2, \ldots$ where M_k simulates M_{k+1}. The sequences and dwell periods were also specified in Definition 2.12. Here, we will use this construction to prove Theorem 1.

8.1 Fault estimation

The theorem says that there is a Turing machine M_1 that can reliably (in the defined sense) simulate any other Turing machine G. Before the simulation starts, the input x of G must be encoded by a code depending on its length $|x|$. We will choose a code that represents the input x as the information content of a pair of cells of M_{k_0} for an appropriate $k_0 = k_0(x)$, and set their kind to Booting. Note that the code does not depend on the length of the computation to be performed. At any stage of

the computation there will be a highest level K such that a generalized Turing machine M_K will be simulated, with its cells of the Booting kind.

As the computation continues and the probability of some fault occurring increases, the encoding level will be raised again and again, by lifting mechanism of Section 4.8. Let $\mathcal{E}_k$ be the event that no burst of level k occurs in the space-time region $[-2B_k, 2B_k) \times [0, T_{k+1})$. This implies that the history on level k can be annotated in this region. Let

$$\mathcal{D}_k = \bigcap_{i<k} \mathcal{E}_i \cap \neg\mathcal{E}_{k+1},$$

then $\mathsf{P}(\bigcap_k \mathcal{E}_k) \geq 1 - \sum_k \mathsf{P}(\mathcal{D}_k)$. Let us show

$$\sum_k \mathsf{P}(\mathcal{D}_k) = O(\varepsilon). \tag{8.1}$$

The number of top level steps on level k is at most $3Q_kF_k$, where F_k is the feathering digression size defined in (4.2). Indeed, in Section 4.8 machine M_k performs at most Q_k simulation steps, but the feathering with big turns may introduce digression steps of up to $2F_k$ per turn.

From Definition 2.12 feathering constant F_k of (4.2) is obtained as

$$F_k = (8k + c_2)^{3+2\rho} < k^4$$

if ρ is a small enough constant and k is large. Hence

$$3F_kQ_k \leq 3c_1^2k^4 \cdot 2^{1.2^k}.$$

Lemma 2.5 bounds the probability of burst of level k by $\varepsilon \cdot 2^{-1.5^{k-1}}$, giving

$$\mathsf{P}(\mathcal{D}_k) = O(\varepsilon k^4 2^{1.2^k - 1.5^{k-1}}),$$

which shows (8.1).

Suppose that machine G produces output at its step t (recall that there is no halting, but the output in cell 0 will not change further). Let t' be any time after the point in the work of machine M_1 at which the simulation of G reaches this stage. For each k let τ_k be the (random) last time before t' when the head of the simulated machine M_k reaches position 0. Let $\mathcal{E}'_k$ be the event that no burst of level k occurs in the space-time region $[-B_k, B_k) \times (\tau_k, \tau_k + T_k]$.

Then $\bigcap_k \mathcal{E}_k \cap \bigcap_k \mathcal{E}'_k$ implies that at time t' the *Output* field of cell 0 has the output of machine G. By an argument equivalent to the one given above, the probability of this event is $1 - O(\varepsilon)$.

Just as above, it is easy to see $\mathsf{P}(\cap_k \mathcal{E}'_k) \geq 1 - O(\varepsilon)$. (Even though the rectangles defining the events $\mathcal{E}'_k$ are random, a probability estimate can be obtained for $\neg\mathcal{E}'_k$ similarly to $\neg\mathcal{E}_k$ above: just average over all possible values of τ_k.)

8.2 Space- and time-redundancy

Even with the simple tripling error-correcting code, there is a constant $c > 1$ such that a colony of level k uses at most c times more space than the amount of information contained in the cell of level $k+1$ that it simulates. Therefore if k is the level that needs to be simulated before an output of G can be reached then the space used at that time is at most c^k times the space needed to just store the information. If G produces output at time t then this is about $c^k t$. The size of cells of level k is of the order of $2^{1.2^k}$, and this is also the the order of the number t of steps of G that can be simulated. So k is about $d \log\log t$ with $d \approx 1/\log 1.2$. This gives a space redundancy factor

$$c^k \approx c^{d \log\log t} = (\log t)^\alpha$$

for some $\alpha > 0$. The estimate for time redundancy is similar.

9 Discussion

A weaker but much simpler solution If our Turing machine could just simulate a 1-dimensional fault-tolerant cellular automaton, it would become fault-tolerant, though compared to a fault-free Turing machine computation of length t, the slow-down could be quadratic. (Such a solution would be only *relatively* simpler, being a reduction to a complex, existing one.) We did not find an easy reduction by just having the simulating Turing machine sweep larger and larger areas of the tape, due to the possibility of the head being trapped too long in some large disorder created by the group of faults. Trapping can be avoided, however, *provided that the length t of the computation is known in advance*. The cellular automaton C can have length t , and we could define a "kind of" Turing machine T with a *circular tape* of size t simulating C. The transition function of T would move the head to the right in every step (with any backward movement just due to faults).

Decreasing the space redundancy We don't know how to reduce the time redundancy significantly, but the space redundancy can be apparently reduced to a multiplicative constant. Following Example 4.3, it is possible to choose an error-correcting code with redundancy that is only a factor δ_k with $\prod_{k=1}^{\infty}(1-\delta_k) > 1/2$. This also requires a more elaborate organization of the computation phase described in Section 4.5 since the total width of all other tracks must be only some δ_k times the width of the *Info* track. For cellular automata, such a mechanism was described in [6].

Other models There is probably a number of models worth exploring with more parallelism than Turing machines, but less than cellular automata: for example having some kind of restriction on the number of active units. On the other hand, a one-tape Turing machine seems to be the simplest computation model for which a reasonable reliability question can be posed, in the framework of transient, non-conspiring faults of constant-bounded probability.

A simpler, universal computation model is the so-called *counter machine*. This has some constant number of nonnegative integer counters (at least two for universality), and an internal state. Each transition can change each counter by ± 1, depends on both the internal state and on the set of those counters with zero value. A fault can change the state and can change the value of any counter by ± 1. It does not seem possible to perform reliable computation on such a machine in any reasonable sense. The statement of such a result cannot be too simple-minded, since there is *some* nontrivial task that such a machine can do: with $2n$ counters, it can remember almost $2n$ bits of information with large probability forever Indeed, let us start the machine with n counters having the value 0, and the other n having some large value (depending on the fault probability ε). The machine will remember forever (with large probability) which set of counters was 0. It works as follows (in the absence of a fault): at any one time, if exactly n values have value 0, then increase each nonzero counter by 1. Otherwise decrease each nonzero counter by 1.

This sort of computation seems close to the limit of what counter machines can do reliably, but how to express and prove this?

Bibliography

[1] Eugene Asarin and Pieter Collins. Noisy turing machines. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming*, pages 1031–1042, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 1

[2] Charles H. Bennett. The thermodynamics of computation – a review. *Intern. J. of Theor. Physics*, 21:905–940, 1981. 1

[3] Ilir Çapuni and Peter Gács. A Turing machine resisting isolated bursts of faults. *Chicago Journal of Theoretical Computer Science*, 2013. See also in arXiv:1203.1335. Extended abstract appeared in SOFSEM 2012. 2.1

[4] Bruno Durand, Andrei E. Romashchenko, and Alexander Kh. Shen. Fixed-point tile sets and their applications. *Journal of Computer and System Sciences*, 78:731–764, 2012. 1

[5] Peter Gács. Reliable computation with cellular automata. *Journal of Computer System Science*, 32(1):15–78, February 1986. Conference version at STOC' 83. 1, 1

[6] Peter Gács. Reliable cellular automata with self-organization. *Journal of Statistical Physics*, 103(1/2):45–267, April 2001. See also arXiv:math/0003117 [math.PR] and the proceedings of STOC '97. 1, 4.3, 9

[7] Peter Gács and John Reif. A simple three-dimensional real-time reliable cellular array. *Journal of Computer and System Sciences*, 36(2):125–147, April 1988. Short version in STOC '85. 1

[8] G. L. Kurdyumov. An example of a nonergodic homogenous one-dimensional random medium with positive transition probabilities. *Soviet Mathematics Doklady*, 19(1):211–214, 1978. 1

[9] Lulu Qian, David Soloveichik, and Erik Winfree. Efficient turing-universal computation with dna polymers. In Yasubumi Sakakibara and Yongli Mi, editors, *DNA Computing and Molecular Programming*, pages 123–140, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 1

[10] Andrei L. Toom. Stable and attractive trajectories in multicomponent systems. In R. L. Dobrushin, editor, *Multicomponent Systems*, volume 6 of *Advances in Probability*, pages 549–575. Dekker, New York, 1980. Translation from Russian. 1

[11] John von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. Shannon and McCarthy, editors, *Automata Studies*. Princeton University Press, Princeton, NJ., 1956. 1